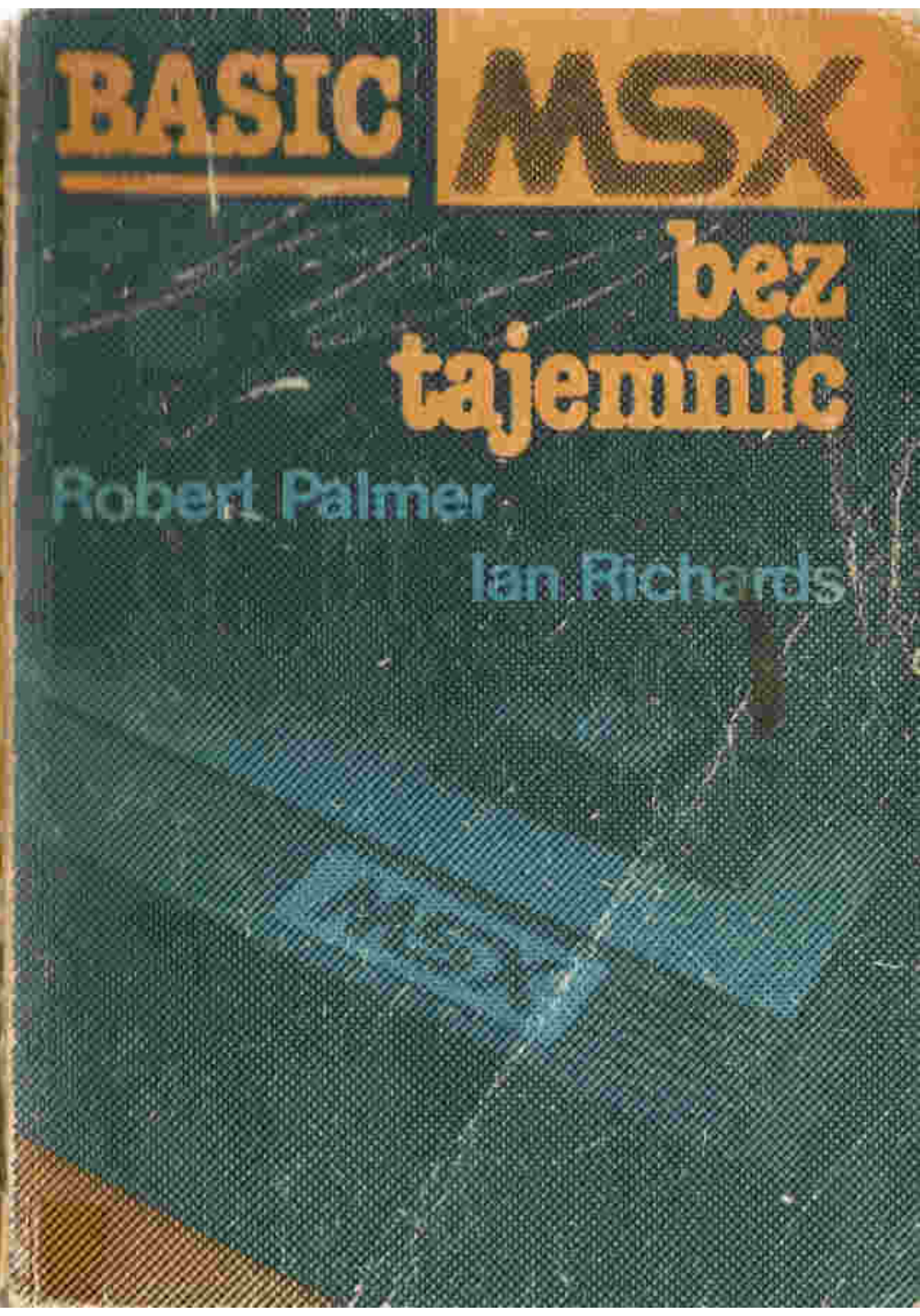
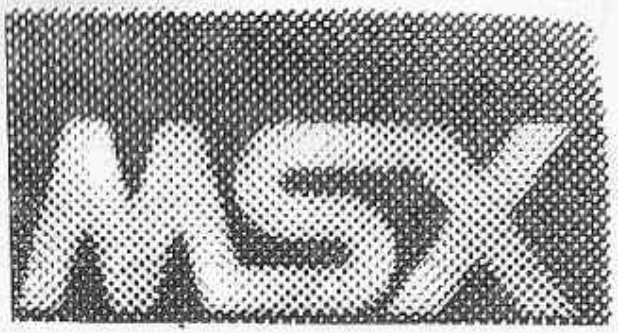
Robert Palmer
Ian Richards

BASIC MSX
bez tajemnic

ISBN 83-214-0652-1

BASIC MSX
bez tajemnic
Robert Palmer
Ian Richards




NGX

Robert Palmer
Ian Richards

BASIC MSX bez tajemnic

Państwowe Wydawnictwo
WIEDZA POWSZECHNA

Tytuł oryginału angielskiego

MSX Basic Revealed

Penguin Books Ltd, Harmondsworth, Middlesex, England

Tłumaczył: *Andrzej Reich*

Okładka: *Marek Zalejski*

Printed in Poland

Redaktor: *Halina Myślicka*

Redaktor techniczny: *Elżbieta Gontarz*

Korektor: *Ewa Garbowska*

ISBN 83-214-0652-1

Wstęp

Pojawienie się komputerów domowych udostępniło milionom ludzi na świecie nowe możliwości życiowe. Od kilku już lat gry komputerowe dostarczają przyjemności i rozrywki, a obecnie mikrokomputery coraz powszechniej używane są do rozwiązywania ważnych zadań w oświacie i znajdują szerokie zastosowanie w gospodarstwie domowym.

Jedną z najbardziej irytujących spraw związanych z tymi urządzeniami jest różnorodność ich rozwiązań konstrukcyjnych. Znaczy to, że program napisany dla komputera danej firmy nie będzie prawdopodobnie możliwy do zrealizowania na komputerze wyprodukowanym przez inną firmę. Nie należy też oczekiwać, by urządzenia zewnętrzne, które zwiększają możliwości danego komputera, mogły współdziałać z innymi modelami wykonanymi nawet w tej samej firmie. Dla właścicieli mikrokomputerów oznacza to — pomijając większe wydatki — ograniczenie wyboru dodatkowego wyposażenia tych urządzeń.

MSX [1] usuwa ten problem, każdy bowiem mikrokomputer zgodny ze standardem MSX zbudowany jest według tych samych założeń, bez względu na producenta. Dodatkowe jego wyposażenie będzie zatem współpracowało z każdym modelem MSX. Dzięki liczbie tych urządzeń komputery MSX ułatwiają życie swym właścicielom.

W książce tej chcemy opowiedzieć, jak do maksimum można

[1] MSX — MicroSoft eXtended — nazwa pochodzi od firmy MicroSoft — MICROcomputer SOFTware, która opracowała tę wersję Basica, zaś *extended* (rozszerzony) informuje, że Basic MSX jest bogatszy od innych wersji tego języka pod względem możliwości (przyp. tłum.).

wykorzystać Basic MSX. Nie jest to jednak po prostu streszczenie instrukcji Basica, ale kurs programowania, który zarówno zupełnym nowicjuszom, jak i doświadczonym programistom pomoże opanować zasady Basica MSX*.

Książka nie wymaga żadnego wstępnego przygotowania. Jej 9 pierwszych rozdziałów wyjaśnia podstawowe pojęcia programowania oraz niektóre ważne elementy konstrukcji Basica. Po tym wprowadzeniu w rozdziale 10 omawiamy projektowanie i tworzenie programów, w 11 — operacje tekstowe, zaś w 12 — operowanie danymi. Standard MSX stwarza duże możliwości operowania dźwiękiem i temu właśnie poświęcony jest rozdział 13. Rozdziały od 14 do 16 omawiają możliwości graficzne, przy czym w ostatnim z nich zwracamy szczególną uwagę na tzw. skrzaty, które są wielkim udogodnieniem przy programowaniu grafiki.

Jesteśmy wdzięczni wielu osobom, które miały udział w napisaniu tej książki — przede wszystkim Cathy Hyde i kolegom z Dragon Data za bezcenną pomoc w początkowym okresie pracy. Dziękujemy Andrewowi Bestowi z Curtis Brown Ltd i Donaldowi McFarlanowi z wydawnictwa „Penguin" za nadanie naszym pomysłom postaci książki. Składamy również podziękowania Martinowi Ottewillowi i Owenowi Williamsowi z X-Data za udostępnienie nam doskonałej drukarki. Szczególnie zaś jesteśmy wdzięczni naszym rodzinom i przyjaciołom, zwłaszcza Meli Richards i Karen Sullivan oraz Nancy Palmer za nieustanne wspomaganie nas kawą i herbatnikami oraz za cierpliwość i wyrozumiałość, bez których książka ta nie zostałaby napisana.

Jeżeli czytelnicy, zapoznając się z tą książką, będą używać równocześnie komputera zarówno do testowania przykładów, jak i wykonywania własnych eksperymentów, zyskają szansę, by zostać biegłymi programistami Basica MSX.

Powodzenia!!!

Ian Richards, Robert Palmer

* Basic MSX — Basic All-purpose Symbolic Instruction Set — symboliczny uniwersalny kod dla szkolenia początkujących (przyp. tłum.).

1

Na czym polega programowanie

Pojawienie się komputerów domowych wywarło w ostatnich latach wyraźny wpływ na życie nas wszystkich. Można obecnie spędzić wieczór walcząc ze smokiem, biorąc udział w wyścigu samochodowym Grand Prix albo broniąc Układu Słonecznego bez ruszania się z fotela. Komputer może też uczyć historii Inków lub pomagać w opanowaniu tajników geometrii. Niewykluczone, że pomoże również w nadzorowaniu interesów albo prowadzeniu budżetu rodzinnego.

Jeden tylko warunek! Jeżeli weźmiecie do ręki jakąkolwiek książkę czy czasopismo poświęcone komputerom, dowiecie się, że powinniście nauczyć się p r o g r a m o w a ć swój mikrokomputer. Komputerowi fanatycy powiedzą, że nie możecie oczekiwać żadnych korzyści płynących z faktu posiadania tego urządzenia, dopóki nie nauczycie się pisać własnych programów. Kursy wieczorowe i kluby komputerowe pełne są zapaleńców zdecydowanych opanować tę wiedzę tajemną. Można wprawdzie kupować programy zapisane na kasetach albo dyskietkach, wprowadzać je do komputera — i już mamy program gotowy do użytku. Cóż więc sprawia, że tysiące ludzi poświęcają swój cenny czas i wkładają tyle wysiłku w naukę programowania komputerów?

Najprostsza odpowiedź brzmi: to po prostu przyjemność. Tradycyjnie uważa się, że programowanie jest trudne i uciążliwe, lecz sytuacja taka należy już do przeszłości. Współczesne komputery są wyposażone w specjalne j ę z y k i, które opracowano po to, by programowanie stało się łatwiejsze. Nawet małe dzieci mogą obecnie poznać urok tworzenia własnych programów. Jest to zawsze wielkie przeżycie — zobaczyć napisany przez siebie program,

uruchomiony po raz pierwszy, bez względu na to, ile się ma lat. Mało które hobby może dać tyle zadowolenia.

Uzyskujecie przy tym większą kontrolę nad swym komputerem. Wprawdzie w sklepach dostępnych jest wiele różnych programów, ale umiejętność samodzielnego programowania pozwala wykorzystać komputer do własnych, specjalnych potrzeb. Może np. przydałby się wam program, który pomógłby prowadzić klub szachowy albo gromadzić przepisy kulinarne? Możecie też wykorzystać komputer do skatalogowania kolekcji znaczków pocztowych lub znalezienia najtańszego sposobu urządzenia mieszkania. A może macie genialny pomysł na nową grę komputerową? W każdym przypadku umiejętność programowania pozwoli wykorzystać mikrokomputer z najlepszym skutkiem. Ponadto, gdy nauczycie się programować, wzbogacicie wiadomości o własnym komputerze i odkryjecie nowe sposoby korzystania z niego.

Opanujecie przy tym umiejętność, która odgrywa coraz większą rolę w świecie. Duże i małe przedsiębiorstwa korzystają z komputerów, aby zwiększyć wydajność pracy. Układy scalone trafiają do wielu urządzeń powszechnego użytku: aparatów fotograficznych, pralek, telewizorów, samochodów itd., a ludzie zaznajomieni z nowymi technologiami będą lepiej przygotowani do korzystania z nich.

Programowanie zatem jest umiejętnością wartą opanowania. Spróbujmy więc poznać lepiej programy komputerowe, a przede wszystkim ustalmy, co rozumiemy przez słowo p r o g r a m. Pojęcie to jest o wiele prostsze, niż można by oczekiwać. Program to nic innego jak ciąg instrukcji. W życiu codziennym najprawdopodobniej korzystacie z wielu programów, nie zdając sobie z tego sprawy. Przyjrzyjmy się np. typowej pralce automatycznej. Takie pralki pracują codziennie w tysiącach domów, a wszystkie wykonują programy, czyli ciągi instrukcji. Pralka może mieć na przykład takie oto dwa programy:

Program 1 Wlej gorącą wodę do najniższego poziomu.
Pierz przez 20 minut.
Wiruj z prędkością 95 obrotów na minutę przez 1 minutę.
Płucz przez 2 minuty.

Wiruj z prędkością 95 obrotów na minutę przez 1 minutę.
Przyspieszaj do prędkości 850 obrotów na minutę w ciągu 25 sekund.
Wiruj przez 2 minuty.

Program 2 Wlej gorącą i zimną wodę do najniższego poziomu.
Podgrzej wodę do 60°C.
Pierz przez 11 minut.
Wiruj z prędkością 95 obrotów na minutę przez 1 minutę.
Płucz przez 8 minut.
Wlej zimną wodę do najniższego poziomu.
Płucz przez 3 minuty.
Wiruj z prędkością 850 obrotów na minutę przez 4 minuty.

Każdy z tych programów jest listą instrukcji, które informują zainstalowany w pralce komputer, jakie czynności mają być kolejno wykonane. Jeżeli wybieramy program 2, pralka zostanie najpierw napełniona ciepłą i zimną wodą. Następna instrukcja zawiera polecenie podgrzania wody do 60°C. Trzecia podaje, że czas prania ma wynosić 11 minut. Instrukcje są wykonywane kolejno, aż do wyczerpania listy. W ten sposób pralka może zrealizować bardzo skomplikowany program. (Dla ścisłości — program podany wyżej byłby zbyt ogólny, aby mógł go zrozumieć komputer zainstalowany w pralce. Musiałby on zostać rozbity na bardziej szczegółowe instrukcje. Niemniej sama zasada pozostaje nie zmieniona.)

Programy nie są zresztą wyłączną domeną komputerów. Pojawiły się one znacznie wcześniej. Przyjrzyjcie się tym przykładom:

A. Składniki: 3 jajka, 20 dkg cukru, 0,5 litra mleka, 2 laski wanilii, 3 łyżki stołowe rumu.
Ubić sztywną pianę z białek dodając szczyptę soli i 6 dkg cukru.
Zagotować 0,25 litra mleka z wanilią. Łyżką stołową nabierać pianę i kłaść na gorące mleko, gotując ją 4 do 5 minut.

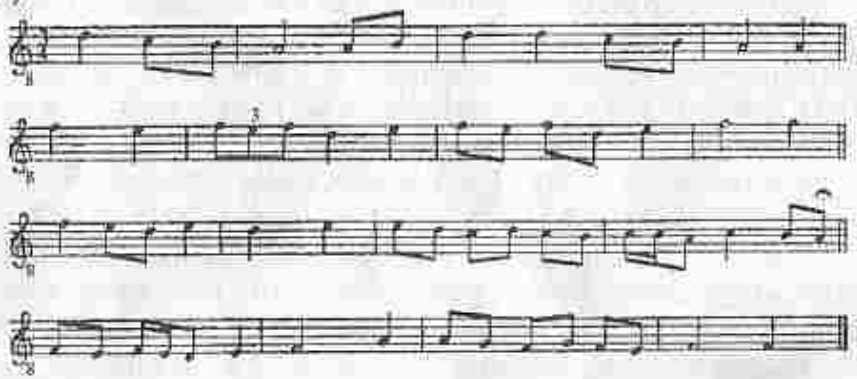
Zrobić krem z pozostałego cukru, mleka i żółtek.
Ostudzonym kremem zalać ugotowaną pianę.

B. Rękawy:
Narzucić 43 oczka na druty nr 12.
1 rząd: b1.1, p.1, * 1.1, powtarzać od * do ostatniego oczka, 1.1.
2 rząd: b1.1, * p.1, 1.1, p.1, powtarzać od * do końca rzędu.
Powtórzyć 1 i 2 rząd 6 razy.
Przełożyć robotę na druty nr 10 i powtarzać wzór od 1 do 18 rzędu.

C. Wyciąć wzdłuż linii kropkowanej.
Założyć A na B.
Podwinąć C pod E.
Założyć D na F.
Założyć G na H.

D.

(*d*)



Rys. 1. Zapis nutowy jest też programem

Gotowanie, roboty na drutach, modelarstwo, muzykę i wiele innych czynności wykonuje się zgodnie z listą szczegółowych instrukcji — programem. Program komputerowy różni się od nich tylko tym, że wydaje polecenia komputerowi, a nie człowiekowi. Spójrzmy na krótki program komputerowy: *

* Polecenia w programach komputerowych są pisane z reguły w języku angielskim. Ich wyjaśnienia podawane są przy omawianiu kolejnych zagadnień oraz zebrane w *Słowniczku* na końcu książki (przyp. red.).


```
10 print „To jest program komputerowy"
20 print „Jest on napisany w języku Basic MSX"
30 end
```

Nie interesuje nas na razie, jak działa ten program. Spróbujcie porównać go z innymi, analogicznymi przykładami.

Książka ta dotyczy programowania w języku o nazwie B a s i c MSX. Aby wyjaśnić, co przez to rozumiemy, musimy powiedzieć nieco więcej o samych komputerach. Konstruowano je początkowo w postaci ogromnych maszyn do „przeżuwania" liczb. Do budowy pierwszych użyto lamp elektronowych — dużych i zawodnych. W rezultacie nawet prymitywny komputer zajmował przestrzeń sporego pokoju. Bardzo trudno było programować te maszyny, a jeszcze trudniej — korzystać z nich. Po wynalezieniu tranzystora zaczęły się one zmniejszać, aż osiągnęły obecne rozmiary domowego mikrokomputera, który nierzadko ma moc obliczeniową większą niż dawna maszyna wielkości pokoju.

Jeżeli trafi się wam sposobność zajrzenia do wnętrza komputera (nie radzimy otwierać własnego — mogłoby to być niebezpieczne i grozi unieważnieniem gwarancji) zobaczycie, że składa się on z garstki małych, prostokątnych „kostek". To właśnie ten miniaturowy układ scalony wykonuje obecnie pracę tysięcy lamp elektronowych stosowanych w pierwszych komputerach.

Najważniejszym układem jest mikroprocesor — mózg komputera, odpowiedzialny za przetwarzanie wszelkich informacji docierających do urządzenia, za wykonanie wszystkich operacji matematycznych i za ogólną kontrolę systemu.

Komputer musi być również w stanie przechowywać informacje, dopóki jest to potrzebne, i w tym celu wykorzystuje dwa rodzaje układów pamięciowych. Pierwszy to RAM — Random Access Memory (pamięć o dostępie swobodnym, pamięć zapisywalna, którą można porównać ze szkolną tablicą). Zapis informacji pozostaje w pamięci RAM póty, póki jest ona potrzebna, a po wykorzystaniu można ją wymazać i na jej miejscu zapisać nową. Jedyna trudność, którą ten typ pamięci stwarza, polega na tym, że jest ona ulotna — po wyłączeniu komputera informacja ginie bezpowrotnie.

Dlatego komputer jest również wyposażony w inny rodzaj pamięci — ROM — Read Only Memory (pamięć stała). Pamięć ROM można porównać do książki, z której odczytuje się zawarte tam informacje, ale nie można zmieniać ich treści. Pamięć ROM nie jest ulotna, zachowuje swoją zawartość nawet po wyłączeniu komputera z sieci.

Układy pamięciowe, procesor i wszystkie inne elementy, które możecie obejrzeć w komputerze (klawiatura, płytka drukowana itd.), noszą nazwę sprzętu (hardware). W tej książce jednak będziemy się zajmowali przede wszystkim programami stosowanymi w komputerach — jest to oprogramowanie (software) systemu.

Mikroprocesor właśnie jest tym elementem komputera, który daje się programować, rozumie specyficzny zestaw poleceń zwany listą rozkazów i potrafi je wykonywać. Dany rozkaz może np. oznaczać polecenie, by procesor pobrał informację z pamięci, inny — żeby wykonał operację dodania dwóch liczb itd. Ten rodzaj programowania nazywa się programowaniem w języku wewnętrznym. Pochłania ono jednak bardzo dużo czasu i jest niezwykle pracochłonne. Konstruktorzy komputerów poradzili z tym sobie, tworząc języki programowania wyższych rzędów, stanowiące zbiór instrukcji, które umożliwiają programiście pisanie programów komputerowych łatwiejsze niż przy użyciu języka wewnętrznego. Aby komputer mógł je zrozumieć, muszą być one jeszcze przetłumaczone na język wewnętrzny. Basic jest właśnie jednym z języków programowania wyższego rzędu. Programy, które piszemy w Basicu, są tłumaczone na język maszynowy przez program zawarty w specjalnym układzie pamięci ROM, który nosi nazwę interpretera Basica.

Basic został opracowany z myślą o pomocy studentom w oswajaniu się z programowaniem, toteż łatwo ten kod opanować. Niestety, stworzono go na początku lat sześćdziesiątych i jego pierwotny opis nie uwzględnia najnowszych osiągnięć techniki komputerowej, takich jak kolor, grafika, dźwięk czy posługiwanie się odbiornikiem telewizyjnym w charakterze monitora. Aby zapełnić te luki, wiele rozszerzonych wersji Basica wyposażono w dodatkowe instrukcje umożliwiające korzystanie z tych osiągnięć. W miarę wzrostu popularności Basica zwiększała się liczba różnych jego „dialektów". Sprawiło to, że większość mikrokomputerów używa dialektów nieco odmiennych, co utrudnia przenoszenie oprogramowania z jednego komputera na inny.

Niniejsza książka dotyczy Basica MSX — dialektu używanego przez wszystkie komputery MSX. Wersja ta jest bardzo prosta w użyciu i zawiera specjalne instrukcje operujące grafiką, dźwiękiem i wszystkimi innymi możliwościami komputera MSX. Pełne zrozumienie języka wprowadzi was w fascynujący świat programowania.

2

Przejdźmy do Basica

Przed przystąpieniem do programowania musimy mieć pewność, że wszystkie elementy naszego komputera są właściwie połączone i pracują prawidłowo. Komputery MSX są tak skonstruowane, aby mogły współpracować z normalnymi domowymi odbiornikami telewizyjnymi. Nieważne, czy telewizor jest duży czy mały, czarno-biały czy kolorowy; wystarczy, by odbierał brytyjski program telewizyjny.[4] Używając odbiornika kolorowego, będziecie mogli oglądać programy w szesnastu barwach generowanych przez komputer. Jeżeli zaś dysponujecie telewizorem czarno-białym, programy będą miały całą gamę odcieni szarości!

Bardzo przydatny jest magnetofon kasetowy. Służy do rejestrowania programów. Można wprowadzić je do komputera i wykonać kiedy indziej. Nie musi to być magnetofon drogi; czasami tańsze magnetofony kasetowe pracują lepiej niż drogi sprzęt klasy hi-fi. Przekonacie się, że mały, przenośny model jest znacznie wygodniejszy w użyciu niż duży magnetofon, ponieważ można go przynieść do komputera (a nie odwrotnie). To samo odnosi się do telewizorów. Użycie dużego, kolorowego telewizora jest bardzo kuszące, ale rodzina nie będzie zachwycona, gdy postanowicie zagrać w wojny gwiezdne podczas jej ulubionego programu telewizyjnego. Spróbujcie znaleźć miejsce, w którym można ustawić komputerowe akcesoria na stałe, gdzie w każdej chwili będą gotowe do użytku. Powinno to być miejsce ciche i spokojne, w którym znajdziecie warunki pozwalające na oswajanie się z komputerem.

Wszystkie niezbędne kable zasilające i połączeniowe powinny znajdować się w opakowaniu razem z komputerem. Jest wśród nich taki, który łączy komputer z gniazdkiem antenowym telewizora, oraz kabel zasilający, który należy wetknąć do gniazdka sieciowego. Komputery MSX mogą wprawdzie korzystać z programów i akcesoriów modeli innych typów, ale sposób połączenia wszystkich elementów nie jest identyczny. Aby uruchomić urządzenie, najlepiej zajrzeć do instrukcji fabrycznej.

Gdy wszystkie elementy komputera zostaną prawidłowo połączone, a on sam włączony do sieci, na ekranie telewizora pojawi się obraz następujący: u góry ekranu — znak firmowy producenta, a u dołu — objaśnienie roli pięciu klawiszy funkcyjnych, które omówimy dalej.

Jeżeli to nie nastąpi, raz jeszcze sprawdźcie wszystko z instrukcją. Skoro na ekranie telewizora nadal nie ma żadnego obrazu, upewnijcie się, czy wszystkie połączenia są w porządku. Nie zapomnijcie też, oczywiście, uruchomić komputer i odbiornik telewizyjny. Jeżeli otrzymacie obraz zdeformowany lub z zakłóceniami, musicie dostroić telewizor.

Po otrzymaniu na ekranie prawidłowego obrazu można rozpoczynać pracę. Z lewej strony ekranu widnieje mały, biały, migający prostokącik. Jest to k u r s o r, a jego zadaniem jest informowanie użytkownika, w którym miejscu ekranu zostanie napisany następny znak. Wszystkie znaki wprowadzamy do komputera posługując się klawiaturą. Jest ona zaprojektowana podobnie do tej, w którą jest wyposażona maszyna do pisania, z tą wszakże różnicą, że klawiatura komputera ma jeszcze pewne dodatkowe klawisze, przeznaczone do wykonywania specjalnych instrukcji Basica.

Spróbujcie nacisnąć jeden klawisz, np. z literą Y. Spójrzcie na telewizor. Litera Y pojawiła się na ekranie tam, gdzie pierwotnie znajdował się kursor, który teraz przesunął się o jedno miejsce w prawo, czyli tam, gdzie pojawi się następny znak.

[4] Większość dostępnych w Polsce mikrokomputerów wyposażona jest w generator obrazu telewizyjnego pracujący w systemie PAL, powszechnie obowiązującym w Europie Zachodniej. Obraz taki można odbierać na używanych w Polsce telewizorach, ale na większości z nich będzie on czarno-biały. Dopiero od niedawna zaczęły się pojawiać w sprzedaży telewizory kolorowe, które są przystosowane do odbioru programu telewizyjnego zarówno w systemach PAL, jak i SECAM. Na tych telewizorach można oczywiście odbierać obraz w kolorze (przyp. tłum.).

W komputerze, podobnie jak w maszynie do pisania, wyświetlane są tylko dolne znaki, czyli małe litery, cyfry itp. Duże litery można pisać naciskając klawisz SHIFT (zmieniacz) równocześnie z klawiszem z wybraną literą. Jeżeli chcecie napisać ciąg dużych liter, musicie nacisnąć klawisz CAPS (blokada zmieniacza). Po jego naciśnięciu na ekranie pojawiać się będą wyłącznie górne znaki (np. duże litery). Aby powrócić do małych liter, należy raz jeszcze nacisnąć klawisz CAPS. Przyjrzyjcie się na przykład klawiszowi z cyfrą 7, powyżej której znajduje się znak &. Jeżeli naciśniecie klawisz 7 jednocześnie z klawiszem SHIFT, na ekranie ukaże się ów znak.

Jeżeli klawisz jest przyciśnięty przez czas dłuższy niż pół sekundy, na ekranie zamiast jednego znaku pojawi się cały ciąg identycznych znaków, które będą wypisywane aż do czasu zwolnienia klawisza. Jest to tak zwana autorepetycja — cecha, jak się później przekonacie, bardzo użyteczna.

Klawisz RETURN (powrót karetki) ma specjalne przeznaczenie. Jego naciśnięcie informuje komputer, że zakończyliście pisanie pewnego tekstu i chcecie, aby komputer zajął się nim. Jeżeli komputer nie rozumie tego, co napisaliście, wypisze komunikat „Syntax error" (błąd składni) albo podobnie brzmiący. Nie ma powodu, by się tym martwić, ponieważ nie jest możliwe uszkodzenie urządzenia jakimkolwiek poleceniem wprowadzanym z klawiatury. Komputer po prostu zignorował to, co napisaliście.

Po wyświetleniu komunikatu o błędzie na ekranie pojawił się napis O.K. Znaczy to, że komputer zakończył wykonywanie zadania (w tym wypadku próbę zinterpretowania napisanego przez was tekstu) i gotów jest obecnie do zajęcia się czymś innym.

Przyjrzyjcie się wreszcie czterem dodatkowym klawiszom umieszczonym na klawiaturze komputera. Oznaczone strzałkami, zgrupowane są w jednym miejscu. Są to tak zwane klawisze sterujące kursorem. Ich podstawowym zadaniem jest przesuwanie kursora po ekranie. Jeżeli naciśniecie klawisz oznaczony strzałką skierowaną w prawo, kursor przesunie się w prawo, naciśnięcie zaś klawisza oznaczonego strzałką skierowaną w lewo spowoduje przesunięcie kursora w lewo. Nie ma chyba potrzeby wyjaśniać, co się stanie gdy naciśnięte zostaną klawisze oznaczone strzałkami skierowanymi w dół i w górę.

Warto poświęcić nieco czasu na oswojenie się z klawiaturą, aby lepiej „poczuć" komputer. Weźcie jakąś książkę lub czasopismo i wpisujcie do komputera zaczerpnięte z nich teksty, aż zapamiętacie położenie różnych klawiszy. Musicie szczególnie uważać, by nie pomylić cyfry jeden i litery I. Musicie też unikać naciskania litery O zamiast cyfry zero. Choć na ekranie niemal nie nie różnią, komputer nie jest w stanie dostrzec tego podobieństwa. Gdy ćwiczycie na klawiaturze zwykłe pisanie tekstu, nie ma to większego znaczenia, ale wkrótce, gdy zaczniecie programować, sytuacja się zmieni. Ćwiczcie też użycie klawiszy SHIFT i CAPS oraz tych ze strzałkami, sterujących położeniem kursora na ekranie, aż uzyskacie pewną wprawę w posługiwaniu się nimi. A gdy już szczęśliwie oswoicie się z klawiaturą, możecie przejść do nauki Basica MSX.

Niektóre instrukcje Basica [*]

Podczas wstępnych wprawek odkryliście już, że wasz komputer ma irytujący zwyczaj wyświetlania komunikatów o błędzie niemal po każdym naciśnięciu klawisza RETURN. Dzieje się tak dlatego, że zasób rozumianych przez niego słów jest bardzo ubogi. Rozumie on tylko tzw. słowa kluczowe należące do Basica MSX i ani jednego więcej. Każde z nich daje maszynie ściśle określone polecenie. Na szczęście dobór słów wykorzystywanych w Basicu MSX jest taki, że można je zestawiać w bardzo wymyślne programy — interesujące i pożyteczne. Zacznijmy jednak od omówienia użycia pojedynczych słów Basica.

Jednym z najistotniejszych zadań komputera jest wypisywanie informacji na ekranie telewizora. Próby rozwiązywania za je-

[*] W zasadzie w programach pisanych w Basicu stosuje się wyłącznie duże litery. Autorzy odeszli nieco od tej zasady, przyjmując następującą konwencję: przykłady użycia instrukcji Basica, stanowiące fragmenty tekstu, pisane są małymi literami, natomiast pełne programy lub ich fragmenty oraz wszelkie instrukcje występujące w tekście pisane są dużymi literami (przyp. tłum.).

go pośrednictwem kluczowych problemów dotyczących życia, wszechświata i wielu jeszcze innych nie bardzo mają sens, jeżeli nie można dotąd otrzymać na nie odpowiedzi. Z tego też powodu jednym z najbardziej uniwersalnych słów w Basicu MSX jest PRINT (drukuj). Wystukajcie na klawiaturze następujące polecenie:

```
print 99
```

Następnie naciśnijcie klawisz RETURN. Na ekranie pojawi się liczba 99. Stało się tak, ponieważ naciśnięcie klawisza RETURN jest sygnałem dla komputera, że macie dla niego pewną pracę do wykonania. Komputer sprawdza następnie napisaną przez was instrukcję i natrafia na polecenie PRINT. Z kolei sprawdza, co ma wydrukować, i znajduje liczbę 99.

A teraz spróbujcie napisać dużymi literami:

```
PRINT 99    (następnie naciśnijcie klawisz RETURN)
```

Otrzymaliście dokładnie taki sam wynik jak poprzednio. Możemy pisać instrukcje języka Basic zarówno małymi, jak i dużymi literami, a nawet mieszając je. Komputer zawsze zinterpretuje prawidłowo. Musimy natomiast upewnić się, że pisownia jest bezbłędna. Komputery są w pewnych sprawach bardzo dobre, ale nie zrozumieją wydanego im polecenia, jeżeli choć jedna litera albo znak interpunkcyjny znajdzie się nie na swoim miejscu. Większość ludzi bez trudu zrozumie, o co chodzi, w instrukcji:

```
pprint 99    (RETURN)
```

Będzie ona jednak całkowicie niezrozumiała dla komputera, który w odpowiedzi wyświetli informację „Syntax error", co oznacza błąd gramatyczny. Trzeba zatem bardzo uważać podczas wpisywania programu.

Ponieważ słowo PRINT wydaje komputerowi polecenie wykonania pewnej czynności, nazywa się ono i n s t r u k c j ą. Możemy użyć go do napisania na ekranie niemal wszystkiego. Spróbujcie wykonać poniższy przykład:

```
print 1000
print −12
```



```
print 23.76 [8]
print 10+8
```

Nie zapomnijcie o naciśnięciu klawisza RETURN po napisaniu każdego polecenia. Wynik otrzymany w ostatnim przykładzie jest dosyć interesujący. Komputer, zamiast wyświetlić na ekranie tekst 10+8, wyliczył i pokazał wartość 10+8. W ten sposób można używać komputera do wykonania wszelkiego rodzaju rachunków.

```
print 100+84.86
print 3876−423
print 7*8
print 25/5
```

Pierwsze dwa przykłady to — odpowiednio — dodawanie i odejmowanie, ale zapis dwóch pozostałych może zaintrygować. Komputery używają bowiem symbolu * dla oznaczenia mnożenia, a symbolu / dla dzielenia. Może to być nieco mylące, ale wkrótce oswoicie się z tym zapisem.

Możliwość przeprowadzania obliczeń w instrukcji PRINT jest bardzo dogodna; a gdybyśmy tak chcieli napisać na ekranie wyrażenie 10+8? Przekonaliśmy się już, że nie można zrobić tego pisząc:

```
print 10+8
```

Aby otrzymać prawidłowy wynik, musimy ująć nasze wyrażenie w cudzysłowy (SHIFT /), tak jak poniżej:

```
print „10+8"
```

W ten sposób, używając cudzysłowów, możemy napisać na ekranie dowolną kombinację znaków:

```
print „fascynujące"
print „8.#:()!"
print „Być albo nie być"
```

[8] Jesteśmy przyzwyczajeni do oddzielania przecinkiem części ułamkowej w liczbie dziesiętnej. W notacji matematycznej stosowanej w krajach anglosaskich zamiast przecinka dziesiętnego używa się kropki dziesiętnej. A ponieważ języki programowania wywodzą się z tych właśnie krajów, musimy przyzwyczaić się do tego nietypowego dla nas zapisu (przyp. tłum.).

Ciąg znaków ujęty w cudzysłowy nosi nazwę string (tekst). Teksty odgrywają dużą rolę w programowaniu komputerów.

Ekran jest już zapełniony przykładami, które nie są nam dalej potrzebne, wprowadźmy więc nową instrukcję języka Basic, aby zrobić sobie trochę miejsca na ekranie. Słowo CLS (clear screen — czyść ekran) jest instrukcją, która usuwa z ekranu wszystkie napisy i ustawia kursor w tak zwanym początkowym położeniu po lewej stronie ekranu. Napiszcie po prostu:

```
cls   (RETURN)
```

i ekran jest już czysty!

Poświęćmy jeszcze nieco czasu instrukcji PRINT. Basic pozwala na napisanie z pomocą jednej instrukcji PRINT więcej niż jednego tekstu czy liczby, jeśli tylko oddzielimy je średnikami:

```
print „472*28.5";" = "; 472*28.5
```

Komputer natrafia na pierwszy tekst i wypisuje go na ekranie. Następnie napotyka średnik i drugi tekst, który również wypisuje na ekranie, tuż za pierwszym. Wreszcie, na końcu instrukcji, znajduje wyrażenie arytmetyczne, którego wartość wylicza i wypisuje za drugim tekstem, oddzielając je od siebie spacją (odstępem).

Możemy również umieścić odstępy między wypisywanymi łańcuchami, oddzielając je w instrukcji PRINT przecinkami:

```
print „jeden", „dwa", „trzy", „cztery"
```

Do tej pory każda napotkana przez nas instrukcja była wykonywana natychmiast po naciśnięciu klawisza RETURN. Jest to tryb konwersacyjny, bardzo przydatny, gdy chcemy otrzymać jedną lub dwie informacje albo przeprowadzić proste obliczenia. Nie nadaje się jednak do bardziej wymyślnych zadań. W rozdziale 3 dowiemy się, jak połączyć instrukcje w program, który stwarza znacznie większe możliwości.

3

Wariacje na temat

Dowiedzieliśmy się już, że program komputerowy jest ciągiem instrukcji. W rozdziale 2. poznaliśmy niektóre spośród słów kluczowych Basica MSX — słów, które rozumie nasz komputer. Są to instrukcje, których możemy użyć w programie. Wprowadzenie listy instrukcji przez wpisanie w trybie konwersacyjnym ich ciągu jedna po drugiej jest możliwe, ale niewygodne, ponieważ komputer zapomina je natychmiast po wykonaniu. Potrzeba więc takiego sposobu przechowywania instrukcji, abyśmy mogli wywołać je ponownie, gdy zajdzie tego potrzeba. Właśnie dzięki temu komputery są urządzeniami o tak ogromnych możliwościach. W Basicu wystarczy dopisać przed każdą instrukcją numer linii.

```
10 cls
20 print „to jest";
30 print „nasz pierwszy program"
```

Mamy tu prawdziwy program napisany w Basicu MSX. Wprowadzając go do komputera trzeba nacisnąć klawisz RETURN na końcu każdej linii, tak samo jak przy pracy w trybie konwersacyjnym. Zasada ta powinna już wejść wam w krew, wobec czego nie będziemy o niej więcej przypominać. W trakcie pracy programu kolejne instrukcje wykonywane są według kolejności numerów linii. Nie jest istotne, jaki numer przyporządkujemy każdej instrukcji, pod warunkiem że nie będą się powtarzać. Gdybyście przypisali której instrukcji numer użyty już wcześniej, komputer wymaże z pamięci instrukcję poprzednią. Nasz program mógł równie dobrze zawierać numery 1, 2, 3 albo 27, 43, 129, albo jeszcze inne, ale rozsądnie jest numerować linie co

dziesięć. Ponieważ instrukcje wykonywane są w kolejności numerów, możemy w razie potrzeby wstawić między nie dodatkowe linie. Na przykład dopiszmy jeszcze:

15 print „patrzcie na to!"

Aby obejrzeć teraz nasz program, użyjemy innego słowa kluczowego:

LIST

Po wydaniu tego polecenia zobaczycie na ekranie wszystkie linie programu wypisane zgodnie z kolejnością numerów linii. Zauważcie, że linia 15 została wprowadzona na właściwe miejsce między liniami 10 i 20. Konwencja numerowania linii co dziesięć daje nam zatem większą elastyczność przy programowaniu. Teraz pozostaje już tylko uruchomić program z pomocą innego słowa kluczowego:

RUN

Po wprowadzeniu tego polecenia komputer zaczyna wykonywać program poczynając od linii o najniższym numerze i przechodzi kolejno przez każdą instrukcję.

LIST i RUN to dwa słowa, których będziecie prawdopodobnie używali podczas programowania częściej niż jakichkolwiek innych, a zatem upewnijcie się, czy rozumiecie dobrze ich rolę.

Jest rzeczą nieuniknioną, że podczas pisania zdarzy się wam od czasu do czasu popełnić błędy. W rozdziale 7 nauczymy się poprawiać programy przy użyciu e d y t o r a, ale na razie możecie korzystać z prostszych metod. Jeżeli pomylicie się podczas wprowadzania linii, napiszcie ją jeszcze raz. Jeżeli jednak nie zauważyliście, że błąd został popełniony, komputer uruchomi program, lecz gdy tylko natrafi na pomyłkę, natychmiast przerwie jego wykonywanie. Na ekranie pojawi się komunikat: „Syntax error in line XXX" (błąd składni w linii XXX). Będziecie wtedy musieli przyjrzeć się tej linii i napisać ją jeszcze raz, już poprawnie.

Gdy skończymy korzystać z programu, musimy wymazać go z pamięci, zanim zaczniemy wprowadzać następne. W przeciwnym razie napotkamy wiele trudności, których źródłem będą fragmenty starego programu wpisane pomiędzy linie nowego. Rzecz jasna,

może to dać przedziwne i zaskakujące efekty. Możemy więc wymazać program z pamięci używając instrukcji:

NEW

Komputer jest teraz gotów na przyjęcie zupełnie nowego programu.

Zmienne

W dotychczasowych przykładach zadowalaliśmy się użyciem liczb lub tekstów o znanej, stałej wartości, jak np. 4, 12.8, „tablica". Bardzo istotna jest jednak możliwość odwoływania się do wielkości, które — w pewnych warunkach — mogą się zmieniać. Przypuśćmy, że chcemy policzyć, ile będzie nas kosztować kupno pięciu, ośmiu, dziesięciu paczek fig. Jeżeli obecna cena fig wynosi 95 zł za paczkę, nasz program mógłby wyglądać tak:[1]

```
10 cls
20 print „Dwie paczki kosztują"; 95*2
30 print „Osiem paczek kosztuje"; 95*8
40 print „Dziesięć paczek kosztuje"; 95*10
```

Jest to dopóty dobre, dopóki cena fig nie zmienia się. Jeżeli jednak cena wzrośnie np. do 98 złotych, to — aby uwzględnić zmianę — będziemy musieli przepisać program. Gdybyśmy chcieli w dodatku znać cenę piętnastu, dwudziestu czy dwudziestu pięciu paczek, zadanie byłoby jeszcze bardziej skomplikowane. Możemy to uprościć wprowadzając z m i e n n ą, której przypiszemy cenę jednej paczki fig.

Przyjrzyjcie się uważnie następującemu programowi:

```
10 cls
20 let figa = 95
```

[1] Jak dotąd żaden powszechnie dostępny mikrokomputer nie jest wyposażony w generator liter polskiego alfabetu. Dlatego też wszystkie teksty pisane po polsku są mało czytelne (np. zamiast „błąd składni" — „blad skladni", zamiast „pięć" — „piec" itd.). Dlatego też, aby zwiększyć czytelność programów, postanowiłem pisać wszystkie nazwy zmiennych i teksty wydruków z uwzględnieniem polskich liter, choć w tej postaci nie da się ich, oczywiście, wprowadzić do komputera (przyp. tłum.).

```
30 print „Dwie paczki kosztują"; figa*2
40 print „Pięć paczek kosztuje"; figa*5
50 print „Dziesięć paczek kosztuje"; figa*10
```

Jeżeli wykonacie ten program, da on takie same wyniki jak wersja oryginalna. Różnica występuje w linii 20. LET (niech) jest kolejnym słowem używanym przez Basic MSX. Informuje ono komputer, że ma on wprowadzić zmienną o nazwie FIGA, której nada na początku wartość 95. Komputer przeznacza fragment pamięci na zapamiętanie tej wartości. Gdy wykonywana jest linia 30, komputer natrafia ponownie na zmienną FIGA i zagląda do pamięci, aby sprawdzić, jaką wartość ma jej przypisać. Następnie podstawia wartość (95) do linii 30, w której pojawia się słowo FIGA. Dokładnie to samo dzieje się podczas wykonywania linii 40 i 50.

Korzyść z takiego rozwiązania polega na możliwości nadania innej wartości zmiennej, gdy tylko mamy na to ochotę. Gdyby cena paczki fig skoczyła do 98 złotych, wystarczyłoby nam zmienić tylko linię 20:

```
20 let figa = 98
```

Gdy wprowadzicie tę linię, komputer zapomni poprzednią, opatrzoną numerem 20, i zastąpi ją nową wersją. Kiedy uruchomicie program, komputer napotka poprawioną instrukcję LET i zmieni wartość zapisaną w pamięci. Zmienna FIGA ma teraz wartość 98.

Wartość zmiennej możemy zwiększyć lub zmniejszyć w dowolnym miejscu programu. Przypuśćmy, że taniej jest kupować figi nie na paczki, ale w pudełkach po dziesięć paczek. Jeżeli kupujemy dziesięć paczek, cena każdej z nich jest mniejsza, powiedzmy, o 5 zł. Musimy wtedy rozszerzyć program o jeszcze jedną linię:

```
45 let figa = figa-5
```

Poczynając od linii 45 zmienna FIGA będzie miała wartość zmniejszoną o 5. A zatem wynik drukowany w linii 50 związany jest z obniżką ceny.

Zmiennej można nadać niemal dowolną nazwę, z wyjątkiem pewnych ograniczeń. A więc może być ona dowolnej długości, ale komputer rozpoznaje tylko pierwsze dwie litery. Znaczy to, że nie

dostrzeże różnicy między dwiema zmiennymi, których nazwy rozpoczynają się od tej samej pary liter. Jeżeli np. nazwiecie trzy zmienne: linia, liczba i litera, komputer potraktuje je wszystkie jako tę samą zmienną.

Nie można też używać w charakterze nazw zmiennych żadnego ze słów kluczowych Basica MSX ani też nadawać nazw zawierających te słowa. Odnosi się to też do symboli matematycznych i innych znaków, które w Basicu MSX mają specjalne znaczenie. Cyfry mogą występować w nazwach na równi z literami, ale pierwszy znak musi być literą. Oto kilka prawidłowych nazw:

```
z
suma
tom 10
Jan Marzec
```

Z kolei poniższe są nieprawidłowe. Czy wiecie dlaczego?

```
runda
HLK Ltd
list
*raport*
22karaty
```

Runda zawiera słowo RUN, zastrzeżone w Basicu MSX, a zatem jest błędna. HLK Ltd ma w środku niedozwolony odstęp, a LIST jest również słowem zastrzeżonym. Nazwa *raport* rozpoczyna się od znaku, który w Basicu oznacza mnożenie, a 22karaty nie rozpoczynają się od litery.

W pełni dozwolone jest użycie nazw jednoliterowych i wielu ludzi korzysta w swoich programach wyłącznie z takich nazw. Kłopot polega na tym, że bardzo trudno jest śledzić logikę programu pełnego nic nie znaczących nazw, zwłaszcza gdy program jest obszerny. Dlatego zmiennym nadaje się zwykle nazwy dłuższe, kryjące w sobie pewne informacje. Jeżeli np. piszecie program, w którym jedną ze zmiennych jest temperatura, nadajcie jej nazwę TEMPERATURA albo przynajmniej TEMP. Wymaga to wprawdzie trochę więcej pisania i zajmuje nieco więcej pamięci, ale te niedogodności są niczym w porównaniu z korzyścia-

mi płynącymi z czytelności programu. Oszczędzicie sobie wielu godzin frustracji i zdenerwowania przy klawiaturze komputera.

Słowo LET zwane jest instrukcją podstawienia, ponieważ pod nazwę zmiennej podstawia konkretną wartość. Możliwe jednak jest użycie instrukcji podstawienia z pominięciem słowa LET:

```
10 cena = cena+10
```

Jeżeli komputer napotka linię programu, która nie zaczyna się od żadnego ze słów kluczowych, zakłada, że jest to instrukcja podstawienia. Zwróćcie uwagę, że znak = ma inne znaczenie w Basicu niż znak równości w normalnej matematyce. Nie oznacza on, że dwie wartości po obu jego stronach są równe, ale że zmienna po jego lewej stronie ma stać się równa tej po prawej stronie, to znaczy nadaje wartość pierwszej z nich.

Korzystamy też ze zmiennych liczbowych, które operują na liczbach; możemy również używać zmiennych tekstowych, które operują na ciągach znaków. Aby Basic MSX mógł rozróżnić te dwa rodzaje zmiennych, musimy zawsze kończyć nazwę zmiennej tekstowej znakiem $. Jeżeli nazwa zmiennej tekstowej kończy się tym znakiem, zakłada się, że jest to zmienna tekstowa. W przeciwnym razie będzie ona interpretowana jako zmienna liczbowa. Pomijając tę różnicę, wszystkie reguły dotyczące prawidłowości nazw odnoszą się do zmiennych obu typów.

Mamy tu kilka typowych zmiennych tekstowych:

```
n$
instrukcja$
miasto$
punkt5$
```

Nowe wartości mogą być im przypisane w dowolnym miejscu programu — tak samo jak w przypadku zmiennych liczbowych. Przeprowadzenie na zmiennych tekstowych operacji matematycznych, takich jak mnożenie lub dzielenie, nie jest możliwe, ale można wykonać na nich działanie zbliżone do dodawania, które omówimy dokładnie w jednym z dalszych rozdziałów. Pod innymi względami zmienne tekstowe mogą być traktowane tak samo, jak ich odpowiedniki liczbowe.


```
10 rozkaz$ = „Dorzuć drew do ognia!"
20 print rozkaz$
```

Gdy komputer wykonuje linię 20, pod nazwę „rozkaz$" podstawia ciąg znaków z linii 10.

Jedna z korzyści płynących z używania zmiennych tekstowych polega na znacznie łatwiejszym tworzeniu programów, w których wielokrotnie powtarzają się te same fragmenty tekstów. Porównajcie te dwa programy:

A.
```
10 print „Element nr 1 jest zapisany pod adresem 4"
20 print „Element nr 2 jest zapisany pod adresem 23"
30 print „Element nr 3 jest zapisany pod adresem 28"
40 print „Element nr 4 jest zapisany pod adresem 96"
50 print „Element nr 5 jest zapisany pod adresem 34"
60 print „Element nr 6 jest zapisany pod adresem 53"
```

B.
```
10 a$ = „Element nr „:b$ = "jest zapisany pod adresem"
20 print a$;1;b$;4
30 print a$;2;b$;23
40 print a$;3;b$;28
50 print a$;4;b$;96
60 print a$;5;b$;34
70 print a$;6;b$;53
```

Pierwszą rzeczą, która rzuca się w oczy, jest tajemniczy dwukropek w linii 10 programu B. Co więcej, w tej samej linii występują dwie instrukcje; jedna jest podstawieniem pod a$, a druga — pod b$. Pozwala na to ten właśnie dwukropek. W jednej linii możecie napisać dowolną ilość instrukcji języka Basic, oddzielonych od siebie średnikami, uważając jedynie, by całkowita długość linii nie przekroczyła 255 znaków. Powinniście jednak zachować ostrożność pisząc wiele instrukcji w jednej linii, ponieważ gmatwa to treść programu i utrudnia wprowadzenie zmian. Niemniej linie o większej liczbie instrukcji są szczególnie użyteczne przy wprowadzaniu zmiennych w jakiś sposób związanych ze sobą. Skoro a$ i b$ w powyższym przykładzie dotyczą drukowania tego samego zdania, wygodne jest umieszczenie ich obu w tej samej linii programu.

Czy widzicie, ile mniej pracy trzeba włożyć w napisanie programu dzięki użyciu tych dwóch zmiennych tekstowych? Za każdym razem, gdy komputer natrafia na jedną z nich, wprowadza na jej miejsce odpowiedni tekst.

Zanim uruchomimy ostatni w tym rozdziale program, weźcie papier i ołówek i zastanówcie się, czy potraficie przewidzieć treść wydruku. Następnie uruchomcie program i sprawdźcie, na ile się wam to udało. Pojęcie zmiennej jest bardzo ważne w programowaniu komputerów, jeżeli więc są rozbieżności między waszą oceną a odpowiedzią komputera, przeczytajcie ten rozdział ponownie, aż zrozumiecie go w pełni. Jeżeli nie macie już problemów ze zmiennymi, jesteście na najlepszej drodze, by stać się programistami komputerów.

```
10 A = 1:B = 2
20 MIEJSCE$ = „Gdynia"
30 PRINT B;MIEJSCE$;A
40 B = 10:C = 5
50 PRINT MIEJSCE$;C
60 A = B+C
70 MIEJSCE$ = „Kalisz"
80 PRINT C;A;MIEJSCE$
```

4

Trochę ruchu

Jesteśmy już w stanie pisać proste, całkiem użyteczne programiki. Na razie wszystkie informacje potrzebne w danym programie muszą od razu być zawarte w jego treści, a wyniki przy kolejnych jego wykonaniach będą zawsze takie same jak za pierwszym razem. Jeżeli potrafimy wprowadzić dodatkowe informacje do programu w trakcie pracy, jego użyteczność znacznie wzrośnie. W Basicu MSX robimy to za pośrednictwem instrukcji INPUT. Jeżeli program w trakcie wykonania napotka taką instrukcję, zawiesza pracę i czeka, aż użytkownik napisze coś na klawiaturze. Na przykład:

```
10 let a = 10
20 input b
30 print a; b
```

Podczas pracy komputer realizuje najpierw linię 10 i nadaje zmiennej a wartość 10. Następnie w linii 20 natrafia na instrukcję INPUT. Instrukcja ta informuje komputer, by czekał tak długo, aż na klawiaturze zostanie napisana jakaś liczba, której wartość zostanie przypisana zmiennej b. Komputer, aby przypomnieć, że czeka na naszą reakcję, wyświetla na ekranie znak zapytania. Jeżeli napiszecie na przykład liczbę 4, a następnie wciśniecie klawisz RETURN, komputer przypisze zmiennej b wartość 4, po czym przejdzie do dalszego wykonania programu. W naszym przykładzie wypisze wartości a i b.

Ponieważ b jest zmienną liczbową, w odpowiedzi na instrukcję z linii 20 możemy napisać tylko liczbę. Ale instrukcja INPUT może odnosić się także do zmiennej tekstowej, jeżeli użyjemy jej

zamiast zmiennej liczbowej. Spróbujcie przerobić linię 20 następująco:

```
20 input x$
```

Możemy teraz napisać na klawiaturze dowolny ciąg znaków, kończąc go naciśnięciem klawisza RETURN. Tekst ten będzie przypisany zmiennej x$. Powinniście dopisać jeszcze jedną linię — drukowanie x$, by przekonać się, że i taka instrukcja działa dobrze.

Może to być jednak bardzo kłopotliwe: uruchomicie program, po czym ujrzycie na ekranie znak zapytania. Jeżeli nie znacie treści programu, skąd macie wiedzieć, co napisać w odpowiedzi? Możemy rozwiązać ten problem uzupełniając instrukcję INPUT stosowną „podpowiedzią" (prompt). Jest to tekst, który chcemy mieć wypisany na ekranie komputera. Pomaga on użytkownikowi w podaniu właściwej odpowiedzi. Włączmy go do instrukcji INPUT:

```
10 input „Podaj swoje imię"; i$
20 print „Serwus,"; i$; „miło mi cię poznać"
```

Mamy tu proste podpowiedzenie, które nie pozostawia użytkownikowi wątpliwości, co ma napisać. Zwróćcie uwagę na sposób zapisu. Tekst podpowiedziany ujęty jest w cudzysłowy i oddzielony średnikiem od nazwy zmiennej. Nie bójcie się korzystać z tej nowości, ponieważ unika się w ten sposób nieporozumień, a program jest znacznie prostszy w użyciu.

Możemy rozszerzyć instrukcję wejścia tak, by zmieścić w jednej linii więcej niż jedną zmienną:

```
10 input powierzchnia, kraj$
```

Podczas wykonywania tej instrukcji komputer zażąda wprowadzenia wartości dla zmiennej POWIERZCHNIA, drukując jeden znak zapytania. Następnie zażąda wprowadzenia wartości dla zmiennej KRAJ$ pisząc dwa znaki zapytania, przypominające, że jest to wciąż część tej samej instrukcji wejścia.

Podczas korzystania z instrukcji INPUT musicie uważać, by wprowadzić właściwy rodzaj danych. Jeżeli spróbujecie podstawić liczbę pod zmienną tekstową*, komputer zasygnalizuje błąd type mismatch (niedopasowanie typów). Można się przed tym zabezpieczyć korzystając jak najczęściej ze słowa „prompt".

Nasze programy stwarzają teraz o wiele większe możliwości, ponieważ jesteśmy w stanie wprowadzać do nich nowe informacje. Jedyny problem sprowadza się do tego, że całą listę instrukcji możemy przed zakończeniem pracy programu wykonać tylko raz. Gdybyśmy chcieli wprowadzić do programu nowy zestaw informacji, musielibyśmy ponownie go uruchomić. Byłoby znacznie lepiej, gdybyśmy mogli spowodować, że komputer po wykonaniu wszystkich instrukcji „skoczyłby" do początku programu, wznawiając jego wykonanie od pierwszej linii. I rzeczywiście, możemy to zrobić za pomocą instrukcji GOTO.

```
10 print „MSX"
20 goto 10
```

W tym krótkim programie instrukcja GOTO w linii 20 informuje komputer, że ma on skoczyć do linii 10 i kontynuować wykonanie programu od tego miejsca. Zrobiwszy to i napisawszy MSX, komputer przejdzie do linii 20, w której ponownie napotka instrukcję GOTO. W rezultacie na ekranie wypisywany będzie bez przerwy tekst MSX, ponieważ komputer wszedł w nie kończącą się pętlę. Jedynym sposobem zatrzymania go jest równoczesne naciśnięcie klawiszy CTRL (control key — klawisz sterujący) i STOP.

Tak naprawdę możemy używać instrukcji GOTO, aby skoczyć do dowolnej linii programu. Poniższy program drukuje rosnący ciąg liczb zaczynający się od liczby 1:

```
10 liczba = 1
20 print liczba
30 liczba = liczba + 1
40 goto 20
```

Bardziej praktyczny byłby program działający jak zwykły

* Błąd w oryginale. Pod zmienną tekstową można podstawić dowolny ciąg znaków, a więc i liczbę. Nie można natomiast zrobić tego w przeciwną stronę, czyli pod zmienną liczbową podstawić ciągu znaków nie będącego liczbą (przyp. tłum.).

kalkulator. Zacznijmy od prostego przykładu dodawania dwóch liczb:

```
10 CLS
20 INPUT „WCZYTAJ LICZBĘ"; A
30 INPUT „WCZYTAJ DRUGĄ LICZBĘ";B
40 C = A+B
50 PRINT „SUMA TYCH DWÓCH LICZB WYNOSI";C
60 GOTO 20
```

Czy widzicie, w jaki sposób instrukcja GOTO zmienia kolejność wykonywania instrukcji? Instrukcja GOTO jest bardzo często używana przez programistów, gdyż daje dużą elastyczność w projektowaniu programów. Jednakże, ponieważ tak chętnie się z niej korzysta, używając jej musicie zachować szczególną ostrożność. Bardzo łatwo możecie doprowadzić do tego, że program stanie się poplątanym i niezrozumiałym ciągiem instrukcji GOTO. Jeżeli do tego dojdzie, stwierdzicie, że nie jest możliwe cofnięcie się i dokonanie zmian ani poprawek, ponieważ nie będziecie w stanie śledzić działania tego programu.

Możemy napisać nasz program dodawania zupełnie bez użycia instrukcji GOTO. Wykorzystamy dwie nowe instrukcje, FOR i NEXT, które mają pomóc w konstruowaniu pętli. Pozwalają one stwierdzić, że pewien zbiór instrukcji ma być wykonywany żądaną ilość razy:

```
10 for licz =1 to 10
20 print licz
30 print licz*100
40 next licz
```

Program ten wypisuje kolejne liczby od 1 do 10 oraz te same liczby pomnożone przez 100. Przyjrzyjmy się mu uważnie. Powtarzane są instrukcje umieszczone między instrukcją FOR i instrukcją NEXT — w naszym przykładzie są to instrukcje w linii 20 i linii 30. Zmienna LICZ w linii 10 jest licznikiem, który zaznacza, ile razy ma być wykonana pętla. Druga część instrukcji FOR (1 to 10) podaje, jaka wartość początkowa ma być nadana zmiennej LICZ oraz jaką wartość końcową powinna ona osiągnąć. Gdy pętla jest wykonywana po raz pierwszy, zmiennej LICZ zostaje nadana wartość 1. Po wykonaniu pętli wartość 1 jest zwiększana do 2 i pętla jest wykonywana ponownie. Następnie LICZ przybiera wartość 3, po czym pętla jest znów wykonywana. Powtarza się to tak długo, aż LICZ osiągnie wartość 10 i pętla zostanie wykonana po raz dziesiąty.

W ten sposób możemy zażądać, aby pętla była wykonywana dowolną, żądaną przez nas ilość razy. Składnia tej instrukcji jest szczególnie wygodna, ponieważ pozwala operować licznikiem (zwanym zmienną sterującą) tak samo jak zwykłą zmienną liczbową. A więc w linii 20 wypisujemy jej bieżącą wartość, a w linii 30 jej wartość pomnożoną przez 100. Przy każdym wykonaniu pętli FOR...NEXT zmienna sterująca zwiększana jest o 1 i w ten sposób zmienia się wypisywany wynik.

Możemy wyznaczyć zakres wartości przybierany przez zmienną sterującą instrukcją FOR. Przepiszcie linię 10 następująco:

```
10 for licz = 5 to 75
```

Przy każdym wykonaniu pętli zmienna sterująca będzie zwiększana o skok równy jedności. Możemy jednak nakazać komputerowi, aby przyjął inną, potrzebną nam wartość skoku:

```
10 for a = 4 to 48 step 4
20 print a
30 next a
```

Możemy nawet spowodować, by komputer, zamiast zwiększać, zmniejszał wartość zmiennej sterującej. Zmieńmy w ostatnim przykładzie linię 10:

```
10 for a = 48 to 4 step -4
```

Czasami wygodnie jest umieścić pętlę w pętli, na co Basic MSX pozwala bez ograniczeń. Ważne jest, by pamiętać, że jedna pętla musi się w całości zawierać w drugiej. A zatem poniższy program jest błędny:

```
10 for z = 1 to 20
20 for y = 5 to 10
30 print z*y
40 next z
50 next y
```

Komputer podejmuje bezładne próby jednoczesnego wykonania dwóch pętli i wyświetla komunikat o błędzie. Program powinien wyglądać tak:

```
10 for z = 1 to 20
20 for y = 5 to 10
30 print z*y
40 next y
50 next z
```

Druga pętla jest obecnie zawarta w pierwszej i można ją wykonać bez przeszkód. Nazywa się to zagnieżdżeniem pętli.

Popatrzmy teraz, w jaki sposób możemy wykorzystać pętlę FOR...NEXT w programie dodającym dwie liczby. Użyjemy jej w najprostszej postaci wyłącznie po to, by pętla została wykonana dokładnie żądaną ilość razy. Zwróćcie uwagę, w jaki sposób użytkownik może ją zdefiniować. Podaje on ilość liczb, które mają być dodane do siebie, a wartość ta użyta jest w instrukcji pętli jako zmienna sterująca: [9]

```
10 CLS
20 INPUT „ILE LICZB CHCESZ DODAĆ?";A
30 FOR B = 1 TO A
40 INPUT „WCZYTAJ LICZBĘ";C
50 D = D+C
60 NEXT B
70 PRINT „SUMA";A; „LICZB WYNOSI";D
```

Nasz komputer może też wykonywać odejmowanie, mnożenie i dzielenie, przepiszmy więc nasz program i zamieńmy go w „potężny" kalkulatorek.

A oto pełny listing (zapis) tego programu:

```
10 CLS
20 PRINT"   MENU"
30 PRINT:PRINT"   1. DODAWANIE"
40 PRINT"   2. ODEJMOWANIE"
50 PRINT"   3. MNOŻENIE"
60 PRINT"   4. DZIELENIE"
70 PRINT:PRINT:INPUT„KTÓRY WARIANT
   WYBIERASZ?";A
80 ON A GOTO 100, 150, 200, 250
90 GOTO 10
100 INPUT „PIERWSZA LICZBA";B
110 INPUT „DRUGA LICZBA";C
120 D = B+C
130 PRINT „SUMA LICZB"; B; „I"; C; „WYNOSI"; D
140 FOR Z = 1 TO 1200: NEXT Z: GOTO 10
150 INPUT „PIERWSZA LICZBA"; B
160 INPUT „DRUGA LICZBA"; C
170 D = B-C
180 PRINT „RÓŻNICA LICZB"; B; „I"; C; „WYNOSI"; D
190 FOR Z = 1 TO 1200: NEXT Z: GOTO 10
200 INPUT „PIERWSZA LICZBA"; B
210 INPUT „DRUGA LICZBA"; C
220 D = B*C
230 PRINT „ILOCZYN LICZB"; B; „I"; C; „WYNOSI"; D
240 FOR Z = 1 TO 1200: NEXT Z: GOTO 10
250 INPUT „PIERWSZA LICZBA"; B
260 INPUT „DRUGA LICZBA"; C
270 D = B/C
280 PRINT „ILORAZ LICZB"; B; „I"; C; „WYNOSI"; D
290 FOR Z = 1 TO 1200: NEXT Z: GOTO 10
```

Przypatrzcie się treści programu i zwróćcie uwagę na zasadę jego działania. Program składa się z czterech zasadniczych części — czterech procedur. Są one bardzo do siebie podobne, ale jedna odpowiada za odejmowanie, druga za mnożenie itd. Jedyną nowością w programie jest instrukcja ON...GOTO w linii 80. Pozwala ona na wykonanie skoku do różnych linii programu, zależnie od wartości zmiennej A. Gdy A równa się 1, komputer wykonuje skok do linii opatrzonej numerem znajdującym się na pierwszym miejscu na liście zawartej w instrukcji ON...GOTO,

[9] Błąd w oryginale. Wczytana zmienna wyznacza liczbę powtórzeń instrukcji umieszczonych w pętli, zaś zmienną sterującą jest zmienna B (przyp. tłum.).

w tym przypadku do linii 100. Gdy A ma wartość 2, komputer skacze do drugiej etykiety na liście — linii 150. Powinniście umieć ustalić, co stanie się, gdy A przybierze wartości 3 i 4.

W tym wypadku użycie instrukcji ON...GOTO jest bardzo wygodne, ponieważ pozwala na włączenie menu do naszego programu. Podobnie jak restauracyjne, menu komputerowe jest spisem pozycji, spośród których możemy wybierać. Wskazując wybraną pozycję z menu informujemy komputer, której z czterech procedur użyjemy. O takim programie mówi się, że jest obsługiwany przez menu.

Przyjrzyjcie się liniom 140, 190, 240 i 290. Wszystkie są jednakowe, ale wyglądają tak, jakby umieszczone w nich instrukcje niczego nie wykonywały. Pomiędzy instrukcjami FOR i NEXT nie ma żadnych instrukcji, wydaje się zatem, że całość pozbawiona jest sensu. Aby zobaczyć, po co zostały użyte, spróbujcie zmienić te linie, zastępując je skróconymi wersjami:

```
140 goto10
190 goto10
240 goto10
290 goto10
```

Jeżeli uruchomicie teraz program, wszystkie procedury zaczną działać, ale... nie będziecie w stanie dojrzeć odpowiedzi! Komputer jest tak szybki, że drukuje wynik i wykonuje skok do instrukcji CLS w linii 10, która czyści ekran, zanim zdążycie przeczytać wynik. Dodajemy zatem pustą pętlę, wprowadzającą opóźnienie w istotnych miejscach programu. Komputer musi w całości wykonać instrukcję pętli, co daje nam dostatecznie dużo czasu, by bez pośpiechu przeczytać wynik, zanim menu powróci na ekran. Pętle opóźniające są bardzo pożyteczne, ponieważ umożliwiają wprowadzenie dowolnie długo trwającego opóźnienia przez zwykłą modyfikację instrukcji FOR...NEXT. Zwróćcie uwagę, jak szybko komputer wykonuje całą, dość długą instrukcję pętli. Daje to pewne wyobrażenie o niezwykłej szybkości jego działania.

5

Zapiszmy to!

Nauczyliśmy się z rozdziału 4 dostatecznie dużo, by móc napisać w Basicu MSX całkiem skomplikowany program. Szkoda tylko, że po napisaniu i sprawdzeniu długiego, złożonego programu traci się go bezpowrotnie w momencie wyłączenia komputera. Program może być ponownie wpisany do komputera, ale jest to zajęcie wielce pracochłonne i zabiera sporo czasu, zwłaszcza w przypadku bardzo długich programów. W tym rozdziale powiemy, w jaki sposób przechować znajdujący się w komputerze program, zapisując go w pamięci trwałej (w tym przypadku jest to kaseta magnetofonowa) i korzystając z pewnych specjalnych instrukcji Basica MSX.

Do zapisania programu na kasecie będą potrzebne:

1. Przewód łączący komputer z magnetofonem kasetowym, należący do wyposażenia komputera.

2. Typowy magnetofon kasetowy.

3. Czysta kaseta.

Można też kupić drogi rejestrator danych przeznaczony do współpracy z komputerami domowymi. Jednakże większość typowych, przenośnych magnetofonów kasetowych pracuje nie gorzej, a może nawet lepiej niż modele znacznie wyższej klasy. Warto też zasilać magnetofon z sieci, a nie z baterii, ponieważ zasilanie sieciowe znacznie rzadziej jest powodem nierównomiernego przesuwu taśmy, będącego przyczyną większości kłopotów związanych z zapisywaniem progamów na taśmie i ładowanie ich z taśmy do komputera.

Łączymy magnetofon z komputerem

Komputery MSX mają różne sposoby podłączania magnetofonu; przeczytajcie zatem uważnie tę część instrukcji dołączonej do komputera, która traktuje o podłączeniu magnetofonu kasetowego i jego obsłudze. Jeżeli napotkacie jakieś trudności, sprawdźcie, czy wszystkie połączenia wykonane są prawidłowo oraz czy kabel łączący komputer z magnetofonem nie jest uszkodzony bądź niesprawny z innego powodu. Gdyby okazało się, że istotnie winien jest kabel, wymieńcie go na nowy tam, gdzie kupiliście komputer.

Gdy magnetofon kasetowy został już prawidłowo połączony z komputerem, włóżcie doń czystą kasetę i przewińcie ją do samego początku. Może się zdarzyć, że magnetofon nie chce przewijać taśmy. Wyjmijcie wtedy z gniazdka wtyczkę, która pozwala na sterowanie pracą magnetofonu z klawiatury komputera. Po przewinięciu taśmy w kasecie wetknijcie ponownie wtyczkę do gniazdka zdalnego sterowania, a następnie wciśnijcie klawisz odtwarzania. Taśma nie będzie się przesuwać. Jeżeli napiszecie teraz na klawiaturze komputera instrukcję MOTOR ON (włącz silnik), magnetofon zacznie pracować. Instrukcja MOTOR ON pozwala w każdej chwili uruchomić magnetofon. Wyłącza się go klawiszem MOTOR OFF.

Teraz, gdy magnetofon pracuje już prawidłowo, musimy mieć coś do zapisania na taśmie. Może ten oto krótki program:

```
40 print „To jest program"
20 print „sprawdzający"
30 print „zapisywanie i odtwarzanie"
40 print „programów dla komputera MSX"
```

Gdy program będzie pracował prawidłowo, wyjmijcie kasetę z magnetofonu i przewijajcie taśmę do przodu, póki nie zniknie biała (lub innego koloru) „rozbiegówka" i ukaże się właściwa, brązowa taśma magnetyczna. Powinniście to zrobić, aby mieć pewność, że początek programu zostanie nagrany, bo taśma rozbiegowa niczego nie rejestruje. Włóżcie następnie kasetę do magnetofonu, po czym naciśnijcie równocześnie klawisze odtwarzania i nagrywania. Napiszcie teraz na klawiaturze komputera:

```
csave„test"
```


Taśma ruszy i będzie się przesuwać przez 2 do 3 sekund, a następnie zatrzyma się. Przez ten czas obraz na ekranie będzie „zamrożony", a potem pojawi się na nim napis O.K. Znaczy to, że ostatnia instrukcja została wykonana prawidłowo; w tym przypadku — program został zapisany na taśmie magnetofonowej. Instrukcja CSAVE jest skrótem od Cassette SAVE (zarejestruj na kasecie), zaś ciąg znaków ujęty w cudzysłowy i znajdujący się po CSAVE jest nazwą programu, który chcemy zapisać.

Gdybyśmy przewinęli kasetę i odtworzyli ją przez głośnik, tak jak normalne nagranie, usłyszelibyśmy tylko piskliwy szum. Są to, stanowiące zapis programu, sygnały wysyłane przez komputer.

Basic MSX pozwala również sprawdzić, czy informacja zapisana na taśmie odpowiada dokładnie informacji znajdującej się w pamięci komputera. Jest to weryfikacja programu. Aby ją przeprowadzić, cofnijcie taśmę w kasecie do początku i naciśnijcie klawisz odtwarzania. Napiszcie następnie:

```
cload?„test"
```

Instrukcja CLOAD jest skrótem od Cassette LOAD (ładowanie z kasety). Ma ona za zadanie wprowadzić zapisany na taśmie program z powrotem do komputera, a ujęta w cudzysłowy nazwa programu, podobnie jak przy instrukcji CSAVE, musi być umieszczona bezpośrednio za słowem CLOAD. Dziwicie się pewnie, dlaczego słowo CLOAD jest w naszym przykładzie zakończone znakiem zapytania. Stanowi on informację dla komputera, że ma porównać program wprowadzany z kasety z programem tkwiącym w pamięci. Jeżeli są one identyczne, komputer poinformuje was, że program został zweryfikowany i nie zawiera błędów. W przeciwnym razie zostanie wyświetlony komunikat o błędzie. Musicie wtedy zapisać program raz jeszcze, w innym miejscu, aby mieć dobrą kopię. Instrukcja VERIFY oszczędzi wam zmartwień i kłopotów spowodowanych błędnie wykonaną instrukcją CSAVE.

Wspomniana wyżej instrukcja CLOAD, przeznaczona do ładowania programów zapisanych na taśmie, zakończona jest zazwyczaj nazwą programu. Nadawanie programom nazw i zaznaczanie ich na kasetach albo na pudełkach od kaset jest praktyką szczególnie zalecaną. Jeżeli w pewnym momencie stwierdzicie, że chcecie

wprowadzić do komputera program bez nazwy albo program, którego nazwę zapomnieliście, cofnijcie taśmę od początku i napiszcie:

cload

Instrukcja CLOAD załaduje wtedy pierwszy napotkany na taśmie program, a kiedy znajdzie się już on w pamięci komputera, na ekranie pojawi się napis O.K.

Jeżeli okaże się, że nie jest to poszukiwany przez was program, będziecie musieli, za pośrednictwem instrukcji CLOAD, wprowadzać kolejno wszystkie znajdujące się na taśmie programy, aż napotkacie ten poszukiwany. Jest to bardzo nużące zajęcie, zwłaszcza gdy na kasecie zapisanych jest wiele programów. Dlatego zawsze lepiej jest nadawać im nazwy, które umieszcza się dodatkowo w notatkach.

Czasami, podczas ładowania programu z taśmy, na ekranie może pojawić się taki oto komunikat:

Device I/O error [18]

Jeżeli istotnie napotkacie taki komunikat, nie wpadajcie w panikę. Znaczy to jedynie, że potencjometry natężenia albo barwy dźwięku ustawione były w złych położeniach. Jeżeli tak się stało, trzeba cofnąć taśmę, zmniejszyć lub zwiększyć poziom natężenia i spróbować raz jeszcze. Po chwili znajdziecie zapewne najlepsze ustawienie potencjometrów. Na przyszłość warto te położenia w jakiś sposób zaznaczyć.

Informacje zawarte w tym rozdziale powinny pozwolić na przechowanie wszystkich ważnych dla was programów dowolnie długo. Na wszelki wypadek — jeszcze cztery dodatkowe rady:

1. Zanim zaczniecie nagrywać upewnijcie się, czy rozbiegówka została przewinięta.
2. Dbajcie o to, by głowice magnetofonu były zawsze czyste.
3. Zawsze weryfikujcie zapisane na taśmie programy.
4. Sporządzajcie zawsze co najmniej dwie kopie programu, a gdy jest on bardzo ważny — więcej.

[18] Błąd urządzenia wejścia/wyjścia (przyp. tłum.).

6

Rozgałęzienia programu

Nasze programy stają się coraz bardziej interesujące. Możemy wypisywać wszystkie kombinacje znaków, zmieniać je, gdy mamy na to ochotę, wykonywać skomplikowane obliczenia w pętlach, wreszcie zapisywać gotowe programy, by je wykorzystać kiedy indziej. W tym rozdziale powiemy o innej ważnej instrukcji Basica: IF...THEN.

Jedną z istotnych właściwości komputerów jest zdolność do podejmowania decyzji. Mogą one być tak zaprogramowane, aby dokonały wyboru między różnymi wariantami, a także, by w wyniku podjętych decyzji wykonały konkretne czynności. Jest wiele urządzeń, które wykonują kolejne instrukcje, oraz takie, które dokonują obliczeń, jednak komputery mają tę właściwość, że w różnych sytuacjach mogą wykonywać różne działania. Cecha ta przemienia je ze zwykłych, przeżuwających liczby maszyn w urządzenia o ogromnych możliwościach. Podejmowaniu decyzji służy w Basicu instrukcja IF...THEN.

```
10 input „Wczytaj liczbę pomiędzy 1 i 5"; liczba
20 if liczba = 3 then print „napisałeś liczbę 3"
30 print „Koniec".
```

W trakcie wykonania linii 20 komputer sprawdza, czy zmienna LICZBA jest równa 3. Jeżeli tak, to instrukcja znajdująca się po słowie THEN jest wykonywana. Jeżeli nie — komputer pomija drugą instrukcję i przechodzi do następnej linii programu. Jest to tak zwana instrukcja warunkowa albo rozgałęzienie programu.

Możemy robić również inne porównania, nie tylko „równa się":

```
10 input „Wczytaj liczbę"; liczba
20 if liczba < 10 then print „Mała liczba"
```

Znak < oznacza m n i e j s z e n i ż. Jest to inny rodzaj porównywania stosowany w instrukcji IF...THEN. Poniżej podajemy pełną listę symboli i ich znaczeń:

= równe
> większe niż
< mniejsze niż
<> nierówne
>= większe lub równe
<= mniejsze lub równe

Następny program korzysta z niektórych spośród tych o p e r a t o r ó w r e l a c j i. Użyte są one do porządkowania zbioru liczb oraz wyznaczenia najmniejszej i największej wartości.

```
10 CLS
20 PRINT „ODLEGŁOŚĆ OD DOMU"
50 INPUT „ODLEGŁOŚĆ OD PIERWSZEGO MIASTA"; A
60 INPUT „ODLEGŁOŚĆ OD DRUGIEGO MIASTA"; B
70 INPUT „ODLEGŁOŚĆ OD TRZECIEGO MIASTA"; C
80 INPUT „ODLEGŁOŚĆ OD CZWARTEGO MIASTA"; D
90 INPUT „ODLEGŁOŚĆ OD PIĄTEGO MIASTA"; E
100 H = A: IF B > A THEN H = B:L = A
105 IF B < A THEN L = B:H = A
110 IF C > H THEN H = C:IF C < L THEN L = C
120 IF D > H THEN H = D:IF D < L THEN L = D
130 IF E > H THEN H = E:IF E < L THEN L = E
140 PRINT „NAJDALSZE MIASTO JEST";H; „KILOMETRÓW STĄD"
150 PRINT „NAJBLIŻSZE MIASTO JEST";L; „KILOMETRÓW STĄD"
160 END
```

Możemy też porównywać teksty. Widać to dobrze w przypadku operatora =, na przykład:

```
10 if a$ = „Jan" then print „To samo"
```

Być może będziecie zaskoczeni faktem, że możemy korzystać również z innych operatorów relacji. Każdy znak używany przez Basic MSX ma swój własny niepowtarzalny numer kodowy. Na

przykład litera A (duża) ma numer kodowy 65. A zatem komputer może porównywać teksty badając kody znaków w każdym z tekstów. Komputer rozpoczyna od porównania pierwszych znaków w obu tekstach. Jeżeli mają one takie same kody, przechodzi do następnej pary itd. Kody te nazywają się kodami ASCII. Słowo ASCII jest akronimem nazwy American Standard Code for Information Interchange (amerykański standardowy kod dla wymiany informacji). Kody te stały się standardem i są używane w wielu komputerach, nie tylko w komputerach MSX. Pełna lista tych kodów jest zamieszczona w *Dodatku C*.

```
10 a$ = „Komputer"
20 b$ = „Kompas"
30 if a$ > b$ then z$ = a$:a$ = b $:b$ = z$
```

W tym programie, jeżeli zmienna a$ jest większa niż zmienna b$, komputer zamienia je miejscami. Używamy zmiennej z$ do chwilowego przechowania a$ podczas operacji zamiany. Gdybyśmy tego nie zrobili, instrukcja a$ = b$ zniszczyłaby zmienną a$. Program z trzema instrukcjami umieszczonymi na końcu linii 30 jest dosyć zagmatwany. Możemy go jednak uprościć posługując się instrukcją SWAP (zamienić):

```
3 if a$ > b$ then swap a$, b$
```

Instrukcja ta przedstawia po prostu miejsca wartości wymienionych zmiennych. Jest ona bardzo przydatna w programach ustawiających elementy w zadanym porządku albo wykonujących podobne czynności.

Instrukcja SWAP nie zawsze jednak jest przydatna. Zdarza się i tak, że po słowie THEN będziecie chcieli umieścić całą grupę instrukcji. Rozwiązaniem jest użycie słowa GOTO w celu wykonania skoku do innej części programu, w której znajdują się interesujące nas instrukcje.

```
10 input „Wczytaj tekst"; a$
20 if a$ > „m" then goto 40
30 goto 10
40 print „tekst";
50 print a$;
```

```
60 print „zaczyna się od litery";
70 print „z drugiej połowy alfabetu"
```

Instrukcja IF...THEN jest tak często używana, że Basic nie wymaga pisania po niej słowa GOTO. Wystarczy, by po słowie THEN dodać właściwy numer linii. Pozwala to zaoszczędzić trochę pisania, tak jak w następnym programie, który korzysta z instrukcji warunkowych do określenia wyniku egzaminu studenta.

```
10 CLS
20 PRINT" OCENY Z EGZAMINU"
30 PRINT: INPUT „NAZWISKO STUDENTA"; A$
40 PRINT: INPUT „OCENA Z MATEMATYKI"; M
50 IF M > 100 OR M < 0 THEN 40
60 IF M > 50 OR M = 50 THEN PRINT „EGZAMIN
   ZDANY"
70 IF M < 50 THEN PRINT „EGZAMIN NIE ZDANY"
80 INPUT „OCENA Z GEOGRAFII"; G
90 IF G > 100 OR G < 0 THEN 80
100 IF G = 50 OR G > 50 THEN PRINT „EGZAMIN
    ZDANY"
110 IF G < 50 THEN PRINT „EGZAMIN NIE ZDANY"
120 INPUT „OCENA Z POLSKIEGO"; P
130 IF P < 0 OR P > 100 THEN 120
140 IF P > 50 OR P = 50 THEN PRINT „EGZAMIN
    ZDANY"
150 IF P < 50 THEN PRINT „EGZAMIN NIE ZDANY"
160 INPUT „OCENA Z FIZYKI"; F
170 IF F < 0 OR F > 100 THEN 160
180 IF F > 50 OR F = 50 THEN PRINT „EGZAMIN
    ZDANY"
190 IF F < 50 THEN PRINT „EGZAMIN NIE ZDANY"
200 H = M+G+P+F
210 H = H/4
220 PRINT „UCZEŃ"; A$; „ŚREDNIA OCENA"; H
230 IF H < 40 THEN PRINT „NIEDOSTATECZNIE"
240 IF H > 39 AND H < 50 THEN PRINT „DOSTATECZ-
    NIE"
250 IF H > 49 AND H < 65 THEN PRINT „DOBRZE"
```



```
260 IF H > 64 AND H < 75 THEN PRINT „BARDZO DO-
    BRZE"
270 IF H > 74 THEN PRINT „CELUJĄCO!! DOSKONA-
    LE!!": END
```

W linii 50 i w kilku innych pojawił się nowy operator: OR (lub). Należy on do grupy operatorów logicznych. Jego nazwa w pełni odpowiada znaczeniu w potocznym języku polskim, bez trudu powinniście więc zrozumieć działanie tego operatora w programie. Aby można było lepiej zrozumieć zasady funkcjonowania operatorów, powiedzmy teraz o relacjach. Są to wyrażenia, które występują w instrukcji warunkowej między IF i THEN (ale można je spotkać nie tylko w tej instrukcji). Mają one zawsze postać:

wartość operator wartość

wartość zaś może być zmienną lub stałą, typu numerycznego lub tekstowego. Na przykład:

```
a > 1
czas$ = „godz 15.00"
długość > = 12
```

Relacja może również zawierać wyrażenia arytmetyczne, np.:

```
a+5 > 12^6
```

Relacja przybiera tylko dwie wartości: true (prawda) lub false (fałsz). Nie istnieje inny wynik. W komputerach MSX obie te wartości są zapisane w postaci liczbowej: zero oznacza fałsz, a jedynka — prawdę. Aby przekonać się, że tak jest w istocie, wykonajcie tych kilka przykładów:

```
print 5 = 6
print 10 > 4
print „liczydło" > „liczenie"
```

Operatory logiczne pozwalają na łączenie relacji.

```
10 if wys > 10 or szer > 20 then print „za duży"
```

Całe wyrażenie będzie miało wartość true, gdy wysokość przekra-

cza 10 albo szerokość wynosi więcej niż 20. Wynik będzie miał wartość true również wtedy, gdy obie relacje będą prawdziwe. Korzystając z operatorów logicznych musicie być bardzo ostrożni i sprawdzać, czy cała instrukcja da wynik zgodny z waszymi przewidywaniami. Możecie ułatwić sobie zadanie odwołując się do tzw. t a b e l e k p r a w d y. Są to tabelki, które — dla danej relacji — podają wyniki wszystkich możliwych kombinacji wartości. Aby wyjaśnić to lepiej, posłużymy się tabelką.

Tabela prawdy operatora OR

X	Y	X or Y
1	1	1
1	0	1
0	1	1
0	0	0

W tym przypadku X i Y mogą być zmiennymi lub wyrażeniami badanymi z pomocą operatora OR. Liczba jeden reprezentuje prawdę, a zero — fałsz. Oto przykład:

```
3 > 2 or 5 < = 12
```

Pierwsze wyrażenie jest prawdziwe, a zatem w kolumnie X wpisujemy 1. Drugie wyrażenie jest również prawdziwe, więc i w kolumnie Y wpisujemy 1. Jeżeli przyjrzymy się tabeli prawdy, stwierdzimy, że naszemu przykładowi odpowiada pierwszy wiersz, czyli:

X	Y	X or Y
1	1	1

Na tej podstawie stwierdzamy, że wynik X or Y jest reprezentowany przez 1. Mówiąc inaczej — wynik jest prawdziwy. W ten sposób możemy badać wynik działania każdego operatora logicznego.

Innym ważnym operatorem jest AND (i). Daje on w wyniku prawdę, jeżeli oba a r g u m e n t y (wartości badane przez operator AND) są prawdziwe.

Tabela prawdy operatora AND

X	Y	X and Y
1	1	1
1	0	0
0	1	0
0	0	0

W poniższym programie, który bada kształt przedmiotów, mamy przykład działania operatora AND.

```
10 input „wysokość"; wysoki
20 input „szerokość"; szeroki
30 input „głębokość"; głęboki
40 if wysoki = szeroki and wysoki = głęboki then 100
50 end
100 print „przedmiot jest sześcianem"
```

W programie umieściliśmy również instrukcję END. Jak można przypuszczać, powoduje ona zatrzymanie programu. W tym przypadku używamy jej, by zapobiec wykonaniu przez komputer linii 100, gdy w linii 40 otrzymamy w wyniku fałsz. Gdybyśmy ją pominęli, linia 100 byłaby wykonywana niezależnie od tego, jakie wartości wprowadzilibyśmy do programu.

Operator OR, o którym mówiliśmy, nosi nazwę „inclusive OR" („l u b" w ł ą c z a j ą c e). Nazwa ta pochodzi stąd, że wynik jest prawdziwy, gdy co najmniej jeden z argumentów jest prawdziwy. Istnieje również XOR — eXclusive OR („l u b" w y ł ą c z a j ą c e), dla którego wynik jest prawdziwy, gdy wyłącznie jeden z argumentów jest prawdziwy. Jest to ważna różnica. Czy widzicie, dlaczego w programie obliczającym punktację w meczu tenisowym musimy użyć XOR, a nie OR?

```
10 input „gracz 1 ma punktów"; p1
20 input „gracz 2 ma punktów"; p2
30 if p1 = 40 xor p2 = 40 then print „piłka setowa"
40 if p1 = 40 and p2 = 40 then print „równowaga"
```

A oto tabela prawdy operatora XOR:

Tabela prawdy operatora XOR

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

Przy użyciu operatora NOT możemy zmienić wynik każdej instrukcji warunkowej na przeciwny. Operator ten zmienia true na false, a false na true. Na przykład:

9 = 12 ma wartość false
not (9 = 12) ma wartość true
8 = 8 ma wartość true
not (8 = 8) ma wartość false

Musicie uważnie korzystać z operatora NOT, ponieważ konstruując wyrażenia logiczne można się łatwo pomylić, na przykład:

not(x = y) and not(x = 2)

da inny wynik niż:

not (x = y and x = 2)

Tabela prawdy operatora NOT

X	not X
1	0
0	1

Istnieją jeszcze dwa operatory logiczne. Jeden z nich daje w wyniku wartość true, gdy oba argumenty mają równocześnie tę samą wartość. Jest to operator EQV.

Tabela prawdy operatora EQV

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

I wreszcie operator IMP. [11]

Tabela prawdy operatora IMP

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

Funkcje i kolejność działań

W dalszej części tego rozdziału powiemy, w jaki jeszcze sposób operować liczbami i wyrażeniami. Przyjrzyjmy się wyrażeniu:

```
print 10*4+3
```

Wydaje się ono całkiem proste; problem polega na tym, że można je dwojako rozumieć. Może to być albo „pomnóż 10 przez 4 i dodaj 3", albo „pomnóż 10 przez sumę 3 plus 4". Obie wersje są możliwe, choć każda z nich daje inny wynik. Aby uniknąć tej niejednoznaczności, w Basicu MSX ustalono kolejność wykonywania działań. Znaczy to, że działaniom matematycznym przypisano różne priorytety. Działanie z wyższym priorytetem wykonywane jest przed działaniem z niższym priorytetem.

Działania arytmetyczne uszeregowane są poniżej według malejących priorytetów — pierwsze z nich ma priorytet najwyższy:

^	potęgowanie
—	negacja
*, /	mnożenie, dzielenie
+, —	dodawanie, odejmowanie

Ponieważ mnożenie ma wyższy priorytet niż dodawanie, mnożenie wykonywane jest w pierwszej kolejności. A zatem w naszym przykładzie komputer zinterpretuje to wyrażenie jako „pomnóż 10 przez 4 i dodaj 3", co daje w wyniku 43.

[11] Operator IMP (IMPlication — wynikanie) daje w wyniku fałsz wtedy i tylko wtedy, gdy z prawdy ma wynikać fałsz (przyp. tłum.).

Pojęcie potęgowania powinno być dla was całkiem jasne. Nie różni się ono niczym od znanego ze szkoły:

5^3 = (5 do potęgi 3) = 5*5*5

16^4 = (16 do potęgi 4) = 16*16*16*16

W Basicu będziemy zapisywali te przykłady następująco:

```
print 5^3; 16^4
```

Czasami musimy obliczyć wartość jakiegoś wyrażenia nie w takiej kolejności, w jakiej zrobiłby to komputer. Przypuśćmy, że chcemy wykonać: „pomnóż 10 przez sumę 4 plus 3". Wiemy, że nie można napisać:

```
print 10*4+3
```

ponieważ to wyrażenie jest wykonywane nie w takiej kolejności, o jaką nam chodzi. Musimy więc ująć w nawiasy to wyrażenie, któremu chcemy nadać najwyższy priorytet:

```
print 10*(4+3)
```

Wyrażenia w nawiasach są zawsze wykonywane w pierwszej kolejności. Dzięki temu możemy sprawić, że komputer wyliczy wartość bardzo skomplikowanego wyrażenia w zadany przez nas sposób. Wolno również umieszczać nawiasy w nawiasach:

```
let a = 4*(6+(12/3))
```

w którym to przypadku wyrażenie w nawiasie wewnętrznym wykonywane jest w pierwszej kolejności.

Basic wyposażony jest również w tak zwane F u n k c j e, które pomagają nam w operowaniu liczbami. Funkcja posiada zawsze nazwę i wykonuje pewne, szczególne operacje na liczbach lub tekstach. Jedna z najbardziej użytecznych funkcji oblicza pierwiastek kwadratowy. Jest to funkcja SQR.

```
print sqr(4)
```

Hokus pokus! Otrzymaliśmy liczbę dwa, która jest pierwiastkiem kwadratowym z czterech. Zauważcie, że liczba, którą pierwiastkujemy, zwana argumentem, ujęta jest w nawiasy. Odnosi się to do

każdej funkcji. Inna funkcja zaokrągla liczbę dziesiętną do liczby całkowitej. Jest to funkcja INT.

```
print int(127.468)
```

INT bada argument i odrzuca jego część ułamkową. Można użyć razem INT i SQR.

```
10 for i = 1 to 1000 step 100
20 print int(sqr(i))
30 next i
```

Powiedzmy teraz o jeszcze jednej funkcji — RND (random — wyrywkowy). Jest to specjalna funkcja, która generuje liczby w sposób losowy. Korzysta się z niej w grach i w tych wszystkich przypadkach, gdy musimy generować liczby losowe. Ostatecznie wojny gwiezdne nie byłyby tak interesującą grą, gdybyście dokładnie wiedzieli, co się za chwilę wydarzy. Funkcja RND generuje liczby losowe z przedziału od 0 do 1. Moglibyście przez chwilę pomyśleć, że w tym przedziale nie ma żadnych liczb, ale w rzeczywistości jest ich tam dużo, np.:

```
0.1363374833334
```

albo

```
0.8919283531246
```

— po prostu są to dwie spośród tysięcy możliwych liczb. Popatrzmy na grupę liczb wygenerowanych przez komputer.

```
10 r = rnd(1)
20 print r
30 goto 10
```

Dlaczego w linii 10 wartość argumentu wynosi 1? Komputer generuje liczby losowe, więc użycie argumentu nie powinno mieć sensu. Argument wpływa jednak na rodzaj generowanych liczb. Komputery MSX generując liczby losowe korzystają z bardzo skomplikowanych wyrażeń matematycznych. Znaczy to, że liczby te nie są w pełni losowe, lecz stanowią część bardzo długiego ciągu liczb, które pozornie nie są ze sobą związane. Ciąg ten jest tak „poplątany", że dla większości zastosowań możemy przyjąć, iż

liczby te są faktycznie losowe. Kłopot polega na tym, że przy każdym uruchomieniu programu otrzymamy zawsze taki sam ciąg liczb losowych. Dzieje się tak, ponieważ komputer za każdym razem w ten sam sposób uruchamia proces generacji.

W zależności od argumentu funkcja RND może być użyta w trzy różne sposoby. Jeżeli jest on większy od zera, przy ponownym wywołaniu wybierany jest kolejny element z ciągu liczb losowych. Gdy jest równy zeru, komputer korzysta z ostatnio wygenerowanej liczby. Jeżeli zaś argument jest mniejszy od zera, komputer wystartuje od elementu z tym właśnie kolejnym numerem.

Uffff! Przebrnęliśmy w tym rozdziale przez ogromną ilość informacji teoretycznych. Czujecie się pewnie tak, jakbyście przyłożyli sobie do głowy rewolwer, a zatem zakończymy rozdział czymś podobnym,[12] czyli programem gry w tzw. rosyjską ruletkę. Teraz, kiedy uporaliście się już z najgorszymi rzeczami związanymi z przetwarzaniem liczb, życie stanie się lżejsze.

```
10 CLS
20 PRINT „RULETKA MSX"
30 PRINT: PRINT „MASZ 6-STRZAŁOWY REWOLWER
   Z JEDNYM NABOJEM"
40 PRINT „JEST ON WYCELOWANY W TWOJĄ LEWĄ
   SKROŃ"
50 PRINT „ABY PRZEŻYĆ, MUSISZ WYBRAĆ 4 LICZBY
   Z ZAKRESU OD 1 DO 6"
60 S = RND(6)
70 INPUT „WCZYTAJ PIERWSZĄ LICZBĘ"; A
80 IF A<1 OR A>6 THEN 70
90 IF A = S THEN GOTO 220 ELSE PRINT „KLIK!"
100 INPUT „WCZYTAJ DRUGĄ LICZBĘ"; B
110 IF B = A OR B<1 OR B>6 THEN 100
120 IF B = S THEN 220 [illegible] PRINT „KLIK!"
130 INPUT „TRZECIA LICZBA"; C
140 IF C = A OR C = B OR C<1 OR C>6 THEN 130
150 IF C = S THEN 220 ELSE PRINT „KLIK!"
```

[12] Uwaga ta dotyczy tylko treści programu (przyp. tłum.).


```
160 INPUT „OSTATNIA LICZBA"; D
170 IF D = A OR D = B OR D = C OR D<1 OR D>6
    THEN 160
180 IF D = S THEN 220 ELSE 190
190 PRINT „ŻYJESZ, ABY ZNOWU WALCZYĆ"
200 INPUT „CZY SPRÓBUJESZ JESZCZE RAZ (T/N)"; C$
210 IF C$ = "T" THEN 10 ELSE END
220 PRINT „BANG!!! MASZ PECHA"
230 GOTO 200
```

Zauważcie, w jaki sposób wyznaczamy w linii 60 odpowiednią liczbę losową. Musimy najpierw pomnożyć liczbę tak, aby znalazła się we właściwym przedziale, a następnie zamienić ją na liczbę całkowitą z pomocą funkcji INT.

Niezwykle istotne jest słowo ELSE, które pojawiło się w linii 90. Jest to rozszerzenie instrukcji IF...THEN, które pozwala wykonać pewną instrukcję, jeżeli warunek logiczny występujący w instrukcji warunkowej nie jest spełniony. W tym przypadku wykonana zostanie instrukcja znajdująca się po słowie ELSE. Rozszerzenie to jest bardzo pomocne przy pisaniu programów w Basicu.

7

Edytor

W miarę wzrostu stopnia skomplikowania i wydłużania programów coraz częściej podczas ich pisania trafiają się nam błędy maszynowe. Uruchamianie programu, gdy ma się wciąż przed oczami owo groźne „Syntax error", jest dosyć irytujące. A jeszcze gorzej, jeżeli błąd występuje w długiej, złożonej linii, ponieważ wtedy trzeba ją całą napisać raz jeszcze. Przynajmniej trzeba by było.

Basic MSX odznacza się szczególną właściwością — umożliwia redagowanie linii programu, czyli poprawianie ich i przebudowywanie, gdyż wyposażony jest w tzw. edytor — specjalny program, który nadzoruje pisanie kolejnych linii. Ten rozdział poświęcamy właśnie edytorowi, a także pewnym bardzo przydatnym instrukcjom.

Bezpośrednio po włączeniu oraz wtedy, gdy żaden program nie jest wykonywany, komputer znajduje się w stanie edycyjnym, co znaczy, że można wtedy redagować dowolną wyświetloną na ekranie linię programu wprowadzając do niej zmiany. Jeżeli przygotujecie krótki program, to wystarczy rozkaz LIST, by została wyświetlona jego treść gotowa do redagowania. Ale gdy program jest długi, najprawdopodobniej nie zmieści się na ekranie w całości. Potrzebna jest więc możliwość wybrania oraz wyświetlenia tej tylko części programu, nad którą będziemy pracować. Można uzupełnić rozkaz LIST tak, aby nam to ułatwił. Istnieje kilka wersji tego uzupełnienia:

list 100 powoduje wyświetlenie linii z numerem 100;

list 100—200 wyświetla wszystkie linie o numerach od 100 do 200;

list 20— wyświetla wszystkie linie od linii z numerem 20 aż do końca programu;

list —340 wyświetla wszystkie linie od początku programu aż do linii z numerem 340;

list. wyświetla ostatnio redagowaną linię.

W ten sposób, z pomocą rozkazu LIST, można wyświetlić na ekranie dowolny fragment programu. Korzystanie z edytora jest bardzo proste. Napiszcie na przykład:

10 print „2 puszki groszku i 1 puszka ananasa"

Przypuśćmy, że chcemy zmienić słowo „groszku" na słowo „gruszek". Pisanie na nowo całej linii tylko po to, aby zmienić w niej jedno słowo, byłoby bardzo żmudne. Naciskając klawisze sterujące kursorem przesuńcie go tak, żeby znalazł się dokładnie pod pierwszą literą słowa „groszku". A teraz napiszcie słowo „gruszek", które — jak zobaczycie — zastąpi słowo „groszku". Naciśnijcie wtedy klawisz RETURN. Po uruchomieniu programu zobaczycie, że komputer zapomniał wszystko to, co przekazaliście mu na temat „groszku". Możliwość zastępowania jednego tekstu drugim bardzo ułatwia redagowanie programów. Aby edytor komputera MSX był bardziej uniwersalny, wyposażono go w inne jeszcze możliwości operowania treścią programu. Przeważnie korzysta się z nich naciskając równocześnie klawisz funkcyjny CTRL (control key) i inny, potrzebny właśnie klawisz. Poniżej omawiamy ich działanie:

CTRL b cofa kursor do początku poprzedniego słowa. Dla edytora słowo jest dowolnym ciągiem znaków nie zawierającym spacji;

CTRL c zatrzymuje pracę programu w trakcie wykonywania instrukcji INPUT (w trakcie oczekiwania na wprowadzenie danych);

CTRL e usuwa część linii znajdującą się na prawo od kursora. Jeżeli linia w programie jest na tyle długa, że na ekranie zajmuje więcej niż jeden wiersz, usunięty zostaje cały tekst znajdujący się poniżej kursora;

CTRL f — przesuwa kursor do początku następnego słowa;

CTRL g — generuje sygnał dźwiękowy. Nie pomaga specjalnie przy redagowaniu programu;

CTRL h — cofa kursor o jedno miejsce, kasując mijany znak;

CTRL i — przesuwa kursor do następnej pozycji tabulatora, wstawiając po drodze spacje. Pozycje tabulatora odległe są od siebie o osiem spacji. Tabulator pomaga w czytelnym redagowaniu programu;

CTRL j — przesuwa kursor na ekranie o jedną linię w dół;

CTRL k — przesuwa kursor do położenia początkowego (lewa strona ekranu);

CTRL l — przesuwa kursor do położenia początkowego oraz czyści ekran;

CTRL m — działa analogicznie do RETURN, czyli informuje komputer, że ma on przystąpić do wykonywania polecenia;

CTRL n — przesuwa kursor na koniec bieżącej linii, dzięki czemu można dopisać do niej kolejny znak;

CTRL r — naciśnięcie tych dwóch klawiszy powoduje przejście do trybu pracy wstawianie (insert mode). Wpisywany wówczas przez nas tekst nie zastępuje tekstu istniejącego, lecz pojawia się w miejscu wskazywanym przez kursor, a tekst znajdujący się na prawo od niego zostaje przesunięty tak, aby dostosować się do tych zmian. Ponowne naciśnięcie CTRL r przywraca poprzedni tryb pracy;

CTRL y — kasuje na ekranie całą linię;

CTRL DEL — usuwa znak wskazywany przez kursor;

Istnieją również kombinacje klawiszy CTRL-litera, które

przesuwają kursor, ale korzystanie z nich nie ma sensu, bowiem znacznie wygodniej jest posługiwać się specjalnie do tego celu przeznaczonymi klawiszami opatrzonymi strzałkami, które sterują kursorem. Na klawiaturze umieszczono wiele klawiszy zastępujących kombinację CTRL-litera. BACK SPACE (cofnij) działa np. tak samo jak CTRL h, a TAB zastępuje CTRL i. HOME/CLS może być użyty zamiast CTRL k oraz CTRL l, a klawisze INS i DEL (delete — usuń) służą do wstawiania i usuwania tekstu, a zatem odpowiadają CTRL r oraz CTRL DEL.

Udogodnienia

Komputery MSX oraz Basic MSX są zatem w stanie wykonać wiele rozkazów dotyczących redagowania tekstów. Dysponują również innymi poleceniami, które mogą ułatwić i przyspieszyć edycję programów. Powiedzmy najpierw o rozkazie DELETE, który pozwala usunąć z programu dowolną linię albo grupę linii. Występuje on w trzech wersjach:

delete 100 — usunie z programu linię 100;

delete —100 — usunie wszystkie linie od początku programu aż do linii o numerze 100;

delete 100—150 — usunie wszystkie linie o numerach od 100 do 150.

Używając rozkazu DELETE należy zachować szczególną ostrożność, bowiem wymazanie efektu czterogodzinnego stukania w klawisze tylko dlatego, że palec trafił przypadkiem nie tam, gdzie trzeba, może przyprawić o atak serca.

Omówimy teraz kolejny rozkaz RENUM (renumber — przenumeruj). Pozwala on zmieniać numery linii całego programu lub jego części. Jest przydatny zwłaszcza, gdy chodzi o wstawienie do programu fragmentu, na który, przy istniejącej numeracji, nie ma miejsca. Stosuje się go również przy porządkowaniu programu przed wydrukowaniem go na drukarce albo zapisaniem na taśmie.

RENUM ma kilka wersji:

renum — zmienia numerację całego programu poczynając od linii 10 ze skokiem 10;

renum 100 — również przenumerowuje cały program, poczynając od pierwszej linii, ale numeracja rozpoczyna się od 100, a nie od 10;

renum 500, 100, 10 — ten rozkaz jest nieco inny. Zmienia bowiem numerację linii programu poczynając nie od pierwszej, ale od setnej. Dawna linia 100 otrzyma numer 500, a kolejna — numery ze skokiem 10. W ten sposób wszystkie linie o numerach mniejszych od 10 pozostaną nie zmienione, reszta zaś programu będzie miała numery linii wzrastające od 500 co 10, czyli 500, 510, 520 itd.;

renum„25 — przypisuje pierwszej linii programu numer 10. Numery dalszych linii będą wzrastały co 25, czyli 10, 35, 60, 85 itd.

Podczas pisania programu można uznać, że własnoręczne numerowanie każdej linii jest dość nużące. Na szczęście w Basicu MSX istnieje rozkaz AUTO (automatycznie), który wykonuje za nas tę pracę.

Po każdym naciśnięciu klawisza RETURN generowany jest kolejny numer linii. AUTO ma cztery wersje:

auto — naciśnięcie klawisza RETURN powoduje, że na ekranie pojawi się numer 10. Po napisaniu linii opatrzonej tym numerem i ponownym naciśnięciu klawisza RETURN na ekranie pojawi się numer 20 itd.;

auto 100 — również będzie generować numery linii, ale poczynając od 100 ze skokiem 10. W ten sposób, przechodząc do pisania

dalszej części programu, możecie pozostawić bez zmiany istniejące już, początkowe linie programu;

auto, 100 — numeracja linii zaczyna się od 0 ze skokiem 100;

auto 100, 100 — numeracja linii zaczyna się od 100 ze skokiem 100, czyli: 100, 200, 300 itd.

Jeżeli rozkaz AUTO wygeneruje numer istniejącej już linii, na ekranie bezpośrednio za numerem wyświetlona zostanie gwiazdka *. Jeżeli np. linia 20 już istnieje, to rozkaz AUTO generując numer 20 wyświetli go jako:

20*

Gdy w linii 20 napiszecie cokolwiek, poprzednia linia 20 zostanie usunięta. Jeżeli jednak chcecie zachować istniejącą linię 20, naciśnijcie RETURN i wtedy pozostanie nie naruszona.

Gdy chcecie zakończyć rozkaz AUTO i wykonać rozkaz RUN, albo LIST, naciśnijcie CTRL. Wrócicie w ten sposób z edytora do Basica.

Możecie czasami chcieć przerwać wykonywanie programu (naciskając CTRL STOP), aby coś w nim sprawdzić lub dokonać pewnych poprawek. Gdybyście chcieli następnie powrócić do wykonywania programu, musicie napisać rozkaz CONT (continue — kontynuować). Rozkazu tego używa się, gdy przerwaliście wykonywanie programu. Napisanie CONT powoduje, że program ruszy od miejsca, w którym nastąpiło przerwanie jego pracy.

Ostatnią rzeczą, którą omówimy, jest rozkaz WIDTH (szerokość), który pozwala ograniczyć liczbę znaków wyświetlanych na ekranie w jednym wierszu. Jeżeli wydacie polecenie:

width 20

a następnie napiszecie jakiś tekst, przekonacie się, że będzie on wyświetlany na ekranie w kolumnie o szerokości 20 znaków.

Warto wspomnieć tu o pewnym rozszerzeniu rozkazu RUN.

Chcąc np. sprawdzić działanie końcówki programu od linii 600, nie trzeba wykonywać go w całości. Wystarczy napisać:

run 600

Komputer zrozumie, że ma rozpocząć wykonywanie programu od linii 600.

Klawisze funkcyjne

Pracując z komputerem MSX zauważyliście zapewne pięć zgrupowanych razem klawiszy oznaczonych symbolami od F1/F6 do F5/F10. Są to klawisze funkcyjne, które umożliwiają szybsze pisanie, ponieważ można je zaprogramować tak, aby odpowiadały pewnym ciągom znaków. Jeżeli przyjrzycie się najniższej linii na ekranie, dostrzeżecie w niej pięć słów: „color", „auto", „goto", „list" i „run". Są to ciągi znaków odpowiadające klawiszom od F1 do F5. Klawiszowi F1 odpowiada „color", F2 — „auto" itd. Jeżeli naciśniecie i przytrzymacie klawisz SHIFT, zauważycie, że słowa u dołu ekranu zmieniły się. Pokazują one teksty przypisane klawiszom z numerami od 6 do 10. Prawdę mówiąc, tekst odpowiadający klawiszowi F6 jest zbyt długi, by mógł w całości być wyświetlony u dołu ekranu, i dlatego widać tam tylko kilka pierwszych znaków. Aby zobaczyć cały tekst, naciśnijcie SHIFT F6. Poniżej znajduje się lista wszystkich tekstów przypisanych tym 10 klawiszom:

F1 color
F2 auto
F3 goto
F4 list
F5 run (return)
F6 color 15, 4, 4 (return)
F7 cload"
F8 cont (return)
F9 list (return), cursor up
F10 (cls) run (return)

Użycie klawisza funkcyjnego daje taki sam efekt jak napisanie odpowiadającego mu tekstu. Na przykład, naciśnięcie F5 jest dokładnie równoważne poleceniu:

run (RETURN)

Są to głównie instrukcje odnoszące się do trybu konwersacyjnego. Po niektórych z nich znajduje się RETURN, co oznacza, że rozkaz zostaje wykonany natychmiast po naciśnięciu danego klawisza, już bez dodatkowego polecenia RETURN. Interesujące są klawisze F9 i F10. F9 pozwala wyświetlić ostatnio napisaną linię programu, a gdy tylko pojawi się na ekranie, kursor wraca natychmiast do jej początku, dzięki czemu redagowanie linii jest łatwiejsze. F10 czyści ekran i powoduje wykonanie programu, który znajduje się właśnie w pamięci.

Klawisze te są bardzo przydatne; co jednak stanie się, jeżeli np. zechcecie umieścić w programie wiele instrukcji wejścia? Wśród tekstów zapisanych pod klawiszami funkcyjnymi nie ma słowa „input" — czyż znaczy to więc, że trzeba je będzie pisać za każdym razem? Nie. Basic MSX pozwala na przypisanie każdemu klawiszowi funkcyjnemu dowolnego tekstu. Gdybyście chcieli na przykład, aby pod klawiszem F2 zamiast tekstu „auto" znajdował się tekst „input", musielibyście napisać:

key2, „input"

Słowo „input" pojawiłoby się u dołu ekranu tam, gdzie uprzednio było wyświetlone słowo „auto". W ten sam sposób można zaprogramować każdy z klawiszy, ale powracają one do stanu początkowego po każdym wyłączeniu komputera z sieci. Aby po instrukcji input dodać RETURN, wystarczy napisać:

key2, „input"+chr$ (13)

Symbol CHR$ (13) jest k o d e m klawisza RETURN i ma takie samo działanie. (Kody znaków zostaną dokładnie omówione w rozdziale 9).

Wprawdzie klawisze te są nam bardzo pomocne podczas pisania programu, ale stałe oglądanie tekstu u dołu ekranu bywa nudne. Można tego uniknąć pisząc:

key off

Tekst u dołu ekranu zniknie. Chcąc wyświetlić go na nowo, trzeba napisać:

key on

Ponieważ mówiliśmy o różnych ułatwieniach przy redagowaniu programu, dobrze byłoby też wspomnieć o pewnym elemencie „stenograficznym" umieszczonym w Basic MSX. Pisząc np. linię:

10 print 100*100

moglibyście zastąpić słowo „print" znakiem ?, a linia w dalszym ciągu będzie poprawna, ponieważ Basic MSX rozumie ten znak. Otrzymamy w ten sposób:

10 ?100*100

Gdy wyświetlicie treść programu, napisany przez was znak zapytania zostanie zamieniony na print, wobec czego program będzie łatwiejszy do zrozumienia. Skrót ten obowiązuje również przy pracy w trybie konwersacyjnym, gdzie jest szczególnie przydatny.

8

Procedury

Przekonacie się wkrótce, że niejednokrotnie zmuszeni będziecie wykonać pewną grupę instrukcji więcej niż jeden raz. Oczywiście, jeżeli grupa ta ma być wykonana kolejno zadaną ilość razy, można wykorzystać pętlę FOR...NEXT, ale nie zawsze! Często potrzebna jest pewna swoboda w odwoływaniu się do tych instrukcji. Można oczywiście napisać pełną ich listę wszędzie tam, gdzie mają być wykonane, ale jest to niepotrzebny wysiłek oraz strata czasu i miejsca w pamięci. Musimy znaleźć sposób, by umieścić te instrukcje w jednym miejscu programu i odwoływać się do nich w razie potrzeby. Przypuśćmy, że chcecie przygotować program, który wypisałby słowa piosenki *Morskie opowieści*. Możemy przygotować program o takiej oto strukturze:

napisz słowa pierwszej zwrotki,
napisz słowa refrenu,
napisz słowa drugiej zwrotki,
napisz słowa refrenu,
napisz słowa trzeciej zwrotki,
napisz słowa refrenu.

Schemat ten można z łatwością rozbudować tak, aby otrzymać działający program, ale duża jego część jest zbędna. Refren powtarza się trzy razy i oczywiście za każdym razem jest taki sam. Wydawcy nut zauważyli to już dawno i opracowali swój własny system skrótów. Jeżeli odnajdziecie w śpiewniku tekst tej piosenki, najprawdopodobniej będzie ona wydrukowana w taki oto sposób:

Kiedy rum zaszumi w głowie,
Cały świat nabiera treści,

Wtedy człowiek chętnie słucha
Morskich opowieści.
(Refren)
Łajba to jest morski statek,
Sztorm to wiatr, co dmucha z gestem,
Cierpi kraj na niedostatek
Morskich opowieści.
(Refren)
Pływał raz marynarz, który
Żywił się wyłącznie pieprzem,
Sypał pieprz do konfitury
I do zupy mlecznej.
(Refren)

Refren: Hej tam, kolejkę nalej,
Hej tam, kielichy wznieśmy,
To zrobi doskonale
Morskim opowieściom.

Refren został wydrukowany tylko raz, na końcu piosenki, natomiast na końcu pierwszej zwrotki znajduje się instrukcja „Refren", która każe nam „skoczyć" do słów refrenu i odśpiewać je. Gdy zrobimy to, wracamy do miejsca, z którego wykonaliśmy skok i zaczynamy śpiewać drugą zwrotkę. Gdy dojdziemy do jej końca, musimy znowu wykonać skok do refrenu. Następnie wracamy tam, skąd wykonaliśmy skok — tym razem na koniec drugiej zwrotki. Ważne jest, żeby pamiętać, z którego miejsca wykonaliśmy skok, w przeciwnym bowiem razie nie bylibyśmy w stanie kontynuować śpiewania od właściwej zwrotki.

Basic wyposażony jest w konstrukcję, która pozwala wykonać podobną sztuczkę w programie. Jest to p r o c e d u r a — część programu, która może być w dowolnej chwili wywołana przez program główny. Zdefiniowanie procedury wymaga użycia dwóch nowych słów. Pierwsze z nich to GOSUB. Poprzedza ono zawsze numer linii:

```
20 gosub 100
```

Pod względem działania instrukcja ta przypomina instrukcję GOTO. Komputer wykonuje skok do linii opatrzonej numerem

znajdującym się po słowie GOSUB (w naszym przykładzie jest to linia 100) i od tego miejsca kontynuuje wykonanie programu. Jest tu jednak istotna różnica. W trakcie wykonywania instrukcji GOSUB komputer zawsze pamięta, w którym miejscu opuścił program główny. Wykonuje on instrukcje zawarte w procedurze aż do momentu napotkania drugiego słowa:

```
110 return
```

Po napotkaniu w procedurze instrukcji RETURN komputer wykonuje powrotny skok do programu głównego, w miejsce, z którego wyszedł do procedury, a następnie, już normalnie, wykonuje dalszy ciąg programu. Poniższy krótki program ilustruje zasady korzystania z procedury:

```
10 print „to jest program główny"
20 gosub 100
30 gosub 200
40 goto 10
100 print „ta procedura jest bez sensu"
110 return
200 print „ta procedura jest również bez sensu"
210 return
```

Teraz przejdźmy już do programu, który drukuje słowa piosenki *Morskie opowieści*. Porównajcie go z pierwotnym tekstem, by przekonać się, jak wygodna jest ta wersja programu:

```
10 CLS
20 A$ = „    MORSKIE OPOWIEŚCI"
30 PRINT A$
40 B$ =" ----------"
50 PRINT B$
60 PRINT
70 PRINT „KIEDY RUM ZASZUMI W GŁOWIE",
80 PRINT: PRINT „CAŁY ŚWIAT NABIERA TREŚCI",
90 PRINT: PRINT „WTEDY CZŁOWIEK CHĘTNIE
   SŁUCHA"
95 PRINT: PRINT „MORSKICH OPOWIEŚCI".
100 GOSUB 240
```

```
110 CLS
120 PRINT A$: PRINT B$
130 PRINT: PRINT „ŁAJBA TO JEST MORSKI STATEK",
140 PRINT: PRINT „SZTORM TO WIATR, CO DMUCHA
    Z GESTEM",
150 PRINT: PRINT „CIERPI KRAJ NA NIEDOSTATEK"
155 PRINT: PRINT „MORSKICH OPOWIEŚCI".
160 GOSUB 240
170 CLS
180 PRINT A$: PRINT B$
190 PRINT: PRINT „PŁYWAŁ RAZ MARYNARZ, KTÓRY"
200 PRINT: PRINT „ŻYWIŁ SIĘ WYŁĄCZNIE PIEPRZEM",
210 PRINT: PRINT „SYPAŁ PIEPRZ DO KONFITURY"
215 PRINT: PRINT „I DO ZUPY MLECZNEJ"
220 GOSUB 240
230 END
240 PRINT: PRINT „REFREN"
250 PRINT„_____"
260 PRINT „HEJ TAM, KOLEJKĘ, NALEJ",
270 PRINT „HEJ TAM, KIELICHY WZNIEŚMY",
280 PRINT „TO ZROBI DOSKONALE"
290 PRINT „MORSKIM OPOWIEŚCIOM".
300 FOR Z = 1 TO 4000: NEXT Z: RETURN
```

Procedura jest bardzo ważną konstrukcją w niemal każdym języku programowania. Sytuacja, w której korzysta z niej więcej niż jeden program, nie jest czymś niezwykłym. Jest to bardzo łatwe — pod warunkiem, że procedura zostanie napisana jako oddzielny podprogram. W takich przypadkach dobrze jest przypisać jej wysokie numery linii, na przykład powyżej 1000. Łatwo wtedy wstawić ją do programu bez przenumerowywania każdej linii. Wielu programistów tworzy zbiory użytecznych procedur, zwane bibliotekami procedur.

We wnętrzu procedury można napisać dowolną instrukcję Basica. Odnosi się to również do GOTO i IF...THEN, a zatem we wnętrzu procedury można wykonywać skoki. Nigdy nie wolno zrobić jednej tylko rzeczy — wykonać skoku poza obszar procedury. Wolno wyjść z niej wyłącznie poprzez instrukcję RETURN.

Złamanie tej reguły prowadzi do zamieszania i nielogiczności w programie. Z drugiej strony, w pełni dozwolone jest tworzenie procedur, w których wywoływane są inne procedury. W takim przypadku komputer śledzi bardzo uważnie położenie punktów powrotu. Gdy w procedurze, która została wywołana przez inną procedurę, komputer napotka instrukcję RETURN, przekazuje sterowanie do pierwszej; gdy w niej napotka instrukcję RETURN, powraca do programu głównego.

W rozdziale 4 mówiliśmy o instrukcji ON...GOTO. Istnieje odpowiednik tej instrukcji, który odnosi się do procedury: ON...GOSUB. Zasada jej działania jest bardzo podobna, tyle tylko, że w tej ostatniej wykonywany jest skok do procedury, a nie do wskazanego miejsca w programie. Procedury te muszą się kończyć instrukcją RETURN. W programach obsługiwanych przez menu lepiej nawet jest stosować instrukcję ON...GOSUB zamiast ON...GOTO. Poniższy program korzysta z procedur przeliczających litry na galony i galony na litry.

```
  5 'WYCZYŚĆ EKRAN
 10 CLS
 15 'WYŚWIETL MENU I TYTUŁ
 20 PRINT „PRZELICZENIE GALONÓW NA LITRY
    I LITRÓW NA GALONY"
 30 PRINT: PRINT „1. GALONY NA LITRY"
 40 PRINT: PRINT „2. LITRY NA GALONY"
 50 PRINT: PRINT „3. KONIEC PROGRAMU"
 55 'WYBIERZ JEDNĄ Z WERSJI
 60 PRINT: INPUT „NAPISZ 1, 2 LUB 3"; WERSJA
 65 'SKOCZ DO WŁAŚCIWEJ PROCEDURY
 70 ON WERSJA GOTO 1000, 2000, 3000
 75 'POWRÓT DO MENU
 80 GOTO 10
1000 CLS
1005 GALONY
1010 INPUT „WCZYTAJ ILOŚĆ GALONÓW"; GAL
1015 'PRZELICZENIE GALONÓW NA LITRY
1020 LIT = GAL*4. 54
1025 'PODANIE ODPOWIEDZI
```

```
1030 PRINT: PRINT „JEST TO ODPOWIEDNIK"; LIT;
     „LITRÓW"
1035 FOR PRZERWA = 1 TO 3000: NEXT PRZERWA
1037  'POWRÓT DO MENU
1040 RETURN
2000 CLS
2005 'LITRY
2010 INPUT „WCZYTAJ ILOŚĆ LITRÓW"; LIT
2015 'PRZELICZENIE LITRÓW NA GALONY
2020 GAL = LIT*0.22
2025 'PODANIE ODPOWIEDZI
2030 PRINT: PRINT „JEST TO ODPOWIEDNIK"! GAL;
     „GALONÓW"
2033 'PRZERWA
2035 FOR PRZERWA = 1 TO 3000: NEXT PRZERWA
2037 'POWRÓT DO MENU
2040 RETURN
3000 CLS
3005 'KONIEC PROGRAMU
3010 END
```

Zauważyliście zapewne nową instrukcję, która wielokrotnie pojawiała się w programie. Szereg instrukcji zaznaczono za pomocą apostrofu ('), który — jak się wydaje — nie wpływa na wykonanie programu. Apostrof zastępuje słowo REM, będące skrótem od „remark" (uwaga); instrukcja ta została wprowadzona po to, by pomóc programiście. Im dłuższy program, tym trudniej zapamiętać, co dzieje się w poszczególnych jego częściach. Instrukcja pozwala wprowadzić do treści programu komentarz, który opisuje wszystkie istotne szczegóły. Komentarze są najbardziej przydatne wtedy, gdy po dłuższej przerwie powraca się do programu w celu wprowadzenia doń pewnych zmian. Za słowem REM można umieścić dowolny ciąg znaków tworzących komentarz. W trakcie wykonywania programu komputer ignoruje całą treść zawartą w instrukcji REM i przechodzi do wykonania kolejnej linii programu. Jedynym zadaniem tej instrukcji jest znalezienie się w treści programu, by pomóc programiście. Jest oczywiście możliwe, że przez całe życie nie napiszecie ani jednego programu zawierającego komentarz, ale w takim przypadku będziecie najprawdopodobniej musieli przeznaczyć wiele czasu na rozwikływanie niezrozumiałych programów. Programy warto pisać tak, by przynajmniej istotna ich część posiadała dokumentację w postaci komentarzy.

Jedna z niezwykłych zalet języka Basic MSX polega na tym, że zawiera on dużą różnorodność rozszerzeń instrukcji ON... ...GOSUB. W tym rozdziale powiemy o dwóch spośród nich. Instrukcja ON STOP...GOSUB sprawdza, czy na klawiaturze nie została naciśnięta kombinacja klawiszy CTRL STOP. Wraz z tą instrukcją używa się kilku słów kluczowych. Oto przykład:

```
10 on stop gosub 100
20 input „on lub off"; wybierz$
30 if wybierz$ = „on" then stop on
40 if wybierz$ = "off" then stop off
50 for i = 1 to 2000: next i
60 goto 20
100 rem procedura pułapka
110 print „naciśnięto CTRL-STOP"
120 return
```

Możemy zdecydować, czy chcemy „schwytać" kombinację CTRL-STOP (to znaczy, w razie jej naciśnięcia, zachować się w szczególny sposób), czy też nie. Możemy też ustalić, jak postąpić w razie jej „złapania". Z linii 10 wynika, że jeżeli po instrukcji STOP ON (pułapka jest aktywna) napotkana zostanie kombinacja CTRL-STOP, nastąpi skok do procedury w linii 100. Instrukcja STOP ON w linii 30 włącza pułapkę, jeżeli napisane zostało „on". Analogicznie instrukcja w linii 40 wyłącza pułapkę. Jeżeli pułapka jest aktywną, komputer bada stan klawiatury, by sprawdzić, czy nie naciśnięto CTRL-STOP. Sprawdzanie to wykonywane jest przed rozpoczęciem każdej instrukcji.

Inna instrukcja, STOP STOP, rejestruje pojawienie się CTRL STOP nawet wtedy, gdy pułapka nie jest aktywna. Jeżeli teraz pojawi się instrukcja STOP ON, komputer natychmiast zareaguje na nią.

Istnieją również instrukcje, które rejestrują naciśnięcie dowolnego klawisza funkcyjnego. Z pomocą instrukcji ON KEY

GOSUB zestawimy listę numerów linii, w których znajdują się procedury rejestrujące naciśnięcie każdego z klawiszy funkcyjnych. Instrukcje KEY(n) ON i KEY(n) OFF pomagają ustalić, które klawisze zostały naciśnięte, a które nie:

```
10 ON KEY GOSUB 100, 200, 300
20 KEY(1) ON
30 KEY(2) ON
40 KEY(3) ON
50 GOTO 50
100 PRINT „został naciśnięty klawisz F1": KEY(1) OFF
110 RETURN
200 PRINT „został naciśnięty klawisz F2": KEY(2) OFF
210 RETURN
300 PRINT „został naciśnięty klawisz F3": KEY(3) OFF
310 RETURN
```

Linia 10 informuje, że w razie wciśnięcia klawisza F1 zostanie wykonany skok do linii 100, jeżeli będzie to klawisz F2 — do linii 200, a jeżeli klawisz F3 — do linii 300. W liniach 20, 30 i 40 zostaje włączone śledzenie tych klawiszy. Natomiast znajdująca się w samej procedurze instrukcja KEY(n) OFF znosi śledzenie tych klawiszy. Uważajcie, by nie pomylić instrukcji KEY ON i KEY(n) ON oraz KEY OFF i KEY(n) OFF.

Ponieważ procedury śledzące różnią się od normalnych, nie zawsze jest konieczne (albo wskazane), by powrócić do tego miejsca w programie, z którego nastąpił skok. Bardzo przydatna natomiast może być możliwość przekazania sterowania do innego miejsca. Można to zrobić dodając stosowny numer linii na końcu instrukcji RETURN, na przykład:

```
return 50
```

przekaże sterowanie do linii 50.

Każdy niemal język komputerowy korzysta w tej czy innej formie z procedur. Zrozumienie zawartych w nich idei będzie bardzo przydatne bez względu na to, jakim komputerem i jakim językiem programowania będziecie się posługiwać w przyszłości.

9
Tablice

W miarę nabywania doświadczenia w programowaniu będziecie coraz częściej napotykali sytuacje, w których występują liczby bądź zmienne w pewien sposób związane ze sobą. Gdybyście np. pisali program, który pamięta urodziny wszystkich waszych krewnych, moglibyście otrzymać całą listę zmiennych. Jedną grupę stanowiłyby te, pod którymi przechowywane by były imiona waszych krewnych, druga zaś grupa zawierałaby daty ich urodzin. Jak dotąd, używaliśmy zmiennych prostych, czyli takich, które w programie nie są bezpośrednio związane z innymi. Programowanie uprościłoby się w istotny sposób, gdyby zebrać zmienne w pewną grupę i nadać im wspólną nazwę. Ułatwiłoby to posługiwanie się różnymi listami i tabelami. Pozostańmy przy zestawieniu dat urodzin waszych krewnych i zastanówmy się, jak byśmy to zrobili bez komputera. Można zbudować taką oto listę:

Krewny (1) 31/08/64
Krewny (2) 10/02/66
Krewny (3) 09/09/50
Krewny (4) 28/02/69

Przypisując każdej pozycji listy nazwę „krewny" dostrzeżemy związek między tymi pozycjami. Wszystkie one są ze sobą spokrewnione. Gdybyśmy na tym poprzestali, musielibyśmy rozwiązać sprawę rozróżniania poszczególnych pozycji, aby móc odwoływać się do dowolnie wybranej, pojedynczej pozycji na liście. Trzeba zatem każdej przypisać numer. Skoro istnieje tylko jedna pozycja o nazwie „Krewny (4)", oznacza to, że w dowolnej chwili możemy odwołać się do niej.

Basic pozwala pracować w dokładnie taki sam sposób. Zamiast zmiennych prostych używamy zmiennych tablicowych. Wprowadzono je do Basica specjalnie po to, by ułatwić operowanie listami i tabelami. Zmienne tablicowe mają dokładnie takie same nazwy, co zmienne proste, ale zawsze występują po nich liczby, które wskazują pojedyncze pozycje tablicy, np.:

```
30 krewny (3) = 090950
```

Z instrukcji tej wynika, że mamy do czynienia z jedną pozycją tablicy o nazwie „krewny", z pewnym szczególnym elementem — numerem 3. Trzeciemu elementowi tablicy „krewny" przypisujemy liczbę 090950. Liczbę, która identyfikuje element tablicy, nazywamy indeksem.

Przy użyciu tablic można otworzyć bardzo długie listy, w których przechowywane są dowolnie wybrane informacje. Z bardzo długimi listami wiąże się jednak pewna niedogodność. Zajmują one mianowicie dużo miejsca pamięci. Dlatego ważne jest, aby komputer wiedział, jak długiej listy może oczekiwać, by zarezerwować dostatecznie duży obszar pamięci do przechowywania wszystkich jej elementów. Trzeba zatem poinformować komputer, jaki maksymalny rozmiar może osiągnąć tablica. Służy temu instrukcja DIM — dimension (wymiar).

```
10 DIM KASETA(25)
```

Powyższa instrukcja rezerwuje miejsce w pamięci dla 25 elementów tablicy „kaseta". Można pominąć instrukcję DIM, jeżeli używa się tablic zawierających nie więcej niż 10 elementów. Komputer zakłada, że tablice bez podanych wymiarów zawierają do 10 elementów. Gdy korzysta się z większych tablic, użycie instrukcji DIM jest konieczne.

Przy korzystaniu z tablic szczególnie istotna jest możliwość używania zmiennej w charakterze indeksu. Znaczy to, że można wydrukować tablicę z pomocą instrukcji pętli FOR...NEXT oraz wykorzystać w pełni właściwości zmiennych. Wyobraźcie sobie, o ile dłuższy byłby poniższy program, gdybyśmy nie korzystali z tablic. Program ten prosi was o wprowadzenie szeregu liczb, a następnie drukuje je w odwrotnej kolejności.


```
10 CLS
20 REM DEKLARACJA TABLICY
30 DIM Z (12)
40 PRINT „WCZYTAJ LICZBY"
50 REM CZYTANIE LICZB W PĘTLI
60 FOR C = 1 TO 12
70 INPUT Z (C)
80 NEXT C
90 FOR C = 12 TO 1 STEP -1
100 PRINT „LICZBA"; C; „="; Z (C)
110 REM DRUKOWANIE LICZB Z PĘTLI
120 NEXT C
```

Istnieją również tablice tekstowe. Korzysta się z nich tak samo jak z tablic liczbowych.

```
10 DIM TEKST$(20)
20 FOR V = 1 TO 20
30 INPUT TEKST$
40 TEKST$(V) = TEKST$
50 NEXT V
60 FOR J = 1 TO 20
70 PRINT TEKST$(J)
80 NEXT J
```

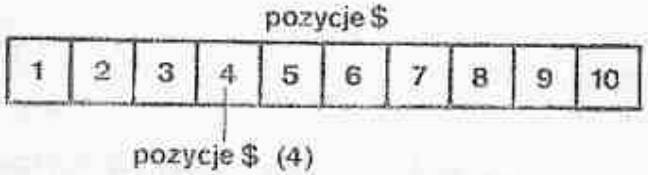
Zwróćcie uwagę, że w tym programie mamy zmienną prostą o nazwie TEKST$ oraz zmienną tablicową o nazwie TEKST$. Komputer MSX rozpoznaje, że są one różnych typów i dlatego nie grozi mu dwukrotne użycie identycznej nazwy. Z drugiej jednak strony może to być mylące, gdy po pewnej przerwie będziecie się przyglądać swym programom. Należy zatem zachować tu pewną ostrożność.

W celu omówienia tablic tekstowych skorzystamy z naszego programu z datami urodzin, który używa tablic tekstowych do zapamiętania właściwych imion:

```
10 REM DEKLARACJA DWÓCH TABLIC
20 DIM A$(5)
```

```
30 DIM A(5)
40 FOR C = 1 TO 5
50 CLS
60 REM WCZYTANIE DO TABLIC IMION I DAT
   URODZIN
70 INPUT „PODAJ IMIĘ"; A$(C)
80 INPUT „PODAJ DATĘ URODZENIA"; A(C)
90 NEXT C
100 CLS
110 PRINT „IMIĘ", „URODZINY"
120 REM DRUKOWANIE IMION I DAT URODZIN
130 FOR C = 1 TO 5 STEP −1
140 PRINT A$(C), A(C)
150 NEXT C
```

Tablic można używać nie tylko po to, aby zapisać listy przedmiotów, lecz również całe tabele. Możemy zbudować dwuwymiarową tabelę, w której zapiszemy nasze dane. Aby to lepiej zrozumieć, popatrzcie na rys. 2. Pokazuje on, że tablicę jednowymia-

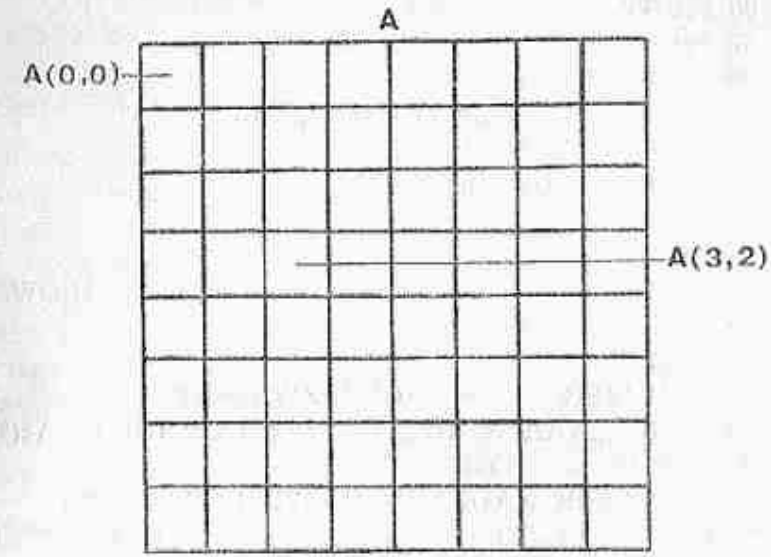


Rys. 2. Jednowymiarowa tablica złożona z 10 elementów

rową można wyobrazić sobie jako zwykły szereg kwadracików (elementów), a każdemu z nich jest jednoznacznie przypisany indeks.

Dwuwymiarowa tablica, która wygląda tak jak na rys. 3, stanowi kolejny krok do przodu.

Mamy zatem całą siatkę elementów, do których możemy się odwoływać. Musimy teraz znaleźć nową metodę oznaczania pojedynczego elementu, ponieważ prosty sposób, dobry dla tablicy jednowymiarowej, tutaj nie wystarcza. Każdemu elementowi



Rys. 3. Tablica dwuwymiarowa

trzeba przypisać dodatkowy indeks. Załóżmy, że nasza tablica nazywa się SIATKA. Gdybyśmy chcieli wypisać na ekranie wartość elementu, który znajduje się w czwartym rzędzie i trzeciej kolumnie, musielibyśmy użyć instrukcji:

```
print siatka (4, 3)
```

Korzystając z tego systemu moglibyśmy wykorzystać nasz komputer do operowania wartościami tablic wszystkich typów. Użyjemy teraz dwuwymiarowej tablicy do zbudowania prostej kartoteki. Pierwszy indeks wskazuje numer karty, do której chcemy się odwołać, drugi zaś odnosi się do konkretnej pozycji na tej karcie. Przeczytajcie uważnie poniższy program, aby dobrze zrozumieć zasadę jego działania.

```
10 REM WYPISANIE MENU
20 CLS
30 PRINT „KARTOTEKA"
40 PRINT: PRINT
50 PRINT „1. WPISANIE NA KARTĘ"
```

```
 60 PRINT „2. CZYTANIE Z KARTY"
 70 PRINT „3. KONIEC PROGRAMU"
 80 REM WYBÓR WERSJI
 90 PRINT: INPUT „WYBIERZ 1, 2 LUB 3"; C
100 ON C GOSUB 130, 270, 390
110 REM POWRÓT DO MENU
120 GOTO 20
130 CLS
140 REM DEKLARACJA TABLICY DWUWYMIAROWEJ
150 DIM A$(5, 20)
160 FOR F = 1 TO 5
170 REM WPROWADZANIE NAZW KART
180 PRINT „KARTA NR"; F; „NAZWA";: INPUT A$(F, 1)
190 FOR EL = 1 TO 4
200 REM WPROWADZANIE POZYCJI NA KARTĘ
210 PRINT „POZYCJA NR"; EL;: POPROSZĘ POZYCJĘ
    NUMER;: INPUT A$(F, EL)
220 NEXT EL
230 CLS
240 NEXT F
250 REM POWRÓT DO MENU
260 RETURN
270 CLS
280 REM NUMER CZYTANEJ KARTY
290 INPUT „KARTA, KTÓRA MA BYĆ PRZECZYTANA"; F
300 REM SPRAWDZANIE POPRAWNOŚCI NUMERU
    KARTY
310 IF F<0 OR F>5 THEN 270
320 FOR EL = 1 TO 4
330 REM WYŚWIETLENIE KARTY NA EKRANIE
340 PRINT A$(F, EL)
350 NEXT EL
360 REM DECYZJA O POWROCIE DO MENU
370 PRINT: INPUT „WRACAĆ DO MENU? (T/N)"; C$
380 IF C$ = „T" THEN RETURN ELSE 370
390 CLS
400 REM KONIEC PROGRAMU
```



```
410 INPUT „KONIEC PROGRAMU? (T/N)"; E$
420 IF E$ = „T" THEN 430 ELSE RETURN
430 CLS
440 END
```

Korzystając z tablic musicie zachować ostrożność, ponieważ wolno je zadeklarować tylko raz. Bardzo łatwo można umieścić w programie instrukcję skoku do tyłu, do pierwotnej instrukcji DIM, co wiąże się z próbą ponownej deklaracji. Jeżeli tak się zdarzy, wystąpi błąd „redimensioned array" (powtórnie zadeklarowana tablica). W zasadzie należy tak zorganizować program, aby tego uniknąć, może się jednak zdarzyć, że będziecie chcieli ponownie zadeklarować tablicę o tej samej nazwie. Zanim jednak to zrobicie, musicie wymazać z pamięci całą jej dotychczasową zawartość. Służy temu instrukcja ERASE (wymaż):

```
erase a$, tabela
```

W powyższym przykładzie wymazane zostałyby tablice a$ i tabela. Można teraz ponownie użyć instrukcji DIM do zadeklarowania tych samych tablic, ale z inymi rozmiarami. Tablice znajdują zastosowanie we wszystkich dziedzinach programowania — od interesów po gry. Następny program jest uproszczoną wersją gry bitwa morska.

Ponieważ gra toczy się na planszy, na której rozmieszczone są okręty wojenne, można ją świetnie opisać z pomocą programu, który korzysta z tablic dwuwymiarowych. W liniach od 10 do 60 wypisywane są na ekranie instrukcje. Linie od 70 do 140 zawierają deklaracje tablic, w których przechowywane są współrzędne okrętów oraz instrukcje przypisujące im wartości losowe. Fragment od linii 150 do 210 odpowiada za wprowadzenie współrzędnych strzału gracza, a w liniach od 220 do 240 porównuje się współrzędne, by sprawdzić, czy strzał był celny. Reszta programu służy obliczaniu punktacji.

```
10 CLS PRINT CHR$(25)
20 REM KOMPOZYCJA OBRAZU NA EKRANIE
30 PRINT „BITWA MORSKA"
40 PRINT: PRINT „NA PLANSZY O ROZMIARACH 8*8
   UKRYTYCH JEST 8 OKRĘTÓW"
```

```
 50 PRINT: PRINT „MUSISZ TRAFIĆ JE W 8 STRZAŁACH"
 60 PRINT: PRINT „POWODZENIA!!"
 70 K = 0
 80 REM DEKLARACJA TABLIC
 90 DIM A(8), B(8), C(8), D(8)
100 FOR J = 1 TO 8
110 REM LOSOWE WYZNACZENIE WSPÓŁRZĘDNYCH OKRĘTÓW
120 A(J) = INT(RND(8))
130 B(J) = INT(RND(8))
140 NEXT J
150 FOR Z = 1 TO 8
160 REM WSPÓŁRZĘDNE STRZAŁU
170 PRINT CHR$ (INT(32*RND(1))+96); „STRZAŁ"; Z
180 INPUT C(Z)
190 INPUT D(Z)
200 PRINT C(Z); „ "; D(Z)
210 FOR J = 1 TO 8
220 REM PORÓWNANIE WSPÓŁRZĘDNYCH STRZAŁU I OKRĘTU
230 IF A(J) = C(Z) AND B(J) = D(Z) THEN K = K+1
240 IF A(J) = C(Z) AND B(J) = D(Z) THEN PRINT „PUNKTACJA"; K
250 REM SPRAWDZANIE WYNIKU
260 IF K = 8 THEN 410
270 REM POWRÓT DO KOLEJNEGO STRZAŁU
280 NEXT J, Z
290 PRINT „CZAS MINĄŁ"
300 REM PODANIE WYNIKU
310 PRINT „TRAFIŁEŚ"; K; „OKRĘTÓW"
320 P = INT ((K*100)/(1+3*RND(1)))
330 PRINT „OCENA STRZELCA"; P
340 REM POŁOŻENIE OKRĘTÓW
350 PRINT „SĄ ONE UKRYTE NA POLACH:-"
360 FOR J = 1 TO 8
370 PRINT A(J); „ "; B(J)
380 NEXT J
```



```
390 REM KONIEC PROGRAMU
400 END
410 PRINT
420 FOR H = 1 TO 7
430 PRINT
440 NEXT H
450 REM JEŻELI WYNIK JEST MAKSYMALNY, PRZEJDŹ DO TEGO FRAGMENTU
460 PRINT „TRAFIŁEŚ JE WSZYSTKIE"
470 FOR U = 1 TO 28
480 PRINT CHR$(INT(32*RND(1))+96);
490 NEXT U
500 GOTO 460
```

W linii 170 pojawiła się nowa funkcja — CHR$. Funkcja ta korzysta z kodów znaków, z którymi zetknęliśmy się niedawno w rozdziale 7. Przypomnijmy, że każdy znak dostępny na komputerach MSX ma swój własny kod, kod ASCII. Funkcja CHR$ zamienia argument na odpowiadający mu znak. Mówiliśmy, że kodem litery A jest 65. Spróbujcie wykonać instrukcję:

```
print chr$(65)
```

Na ekranie pojawi się litera A. Przećwiczcie użycie tej funkcji biorąc argumenty o wartościach od 32 do 127. Są to kody ASCII znaków, które dają się wydrukować. Znaki o kodach mniejszych od 32 mają inne funkcje, takie jak na przykład obsługa drukarki. Kody pomiędzy 128 i 255 odnoszą się do nowego zestawu znaków. Te dodatkowe znaki ułatwiają elegancką kompozycję obrazu i wspomagają operacje graficzne. Pomówimy o tym szczegółowo nieco później. A teraz uruchomcie poniższy program, aby obejrzeć pełen zestaw znaków komputera MSX.

```
10 cls
20 for i = 32 to 255
30 print chr$(i)
40 next i
```

W naszej bitwie morskiej funkcja CHR$ została użyta w celu zwiększenia czytelności obrazu na ekranie.

Istnieje również funkcja odwrotna do CHR$, której działanie jest przeciwne. Funkcja ASC przypisuje znakowi jego kod:

```
print asc(A)
```

Jeżeli argumentem jest znak o długości większej niż jeden znak, to wartość funkcji odpowiada pierwszemu znakowi:

```
print asc(„mnóstwo")
```

Są to tylko dwie spośród funkcji dostępnych w Basicu MSX. Zanim omówimy je szczegółowo, zajmijmy się innym ważnym zagadnieniem programowania — sztuką projektowania programów.

10

Projektowanie programu

Kiedy przychodzi do głowy pomysł jakiegoś programu, aż kusi, aby usiąść przed klawiaturą mikrokomputera i przystąpić do pisania go instrukcja po instrukcji. Myślicie zapewne, że wiecie, jak ma wyglądać ten program i że w związku z tym nie będzie problemów z uruchomieniem go. Popełnić jakiś podstawowy błąd w konstrukcji programu jest jednak bardzo łatwo. Stwierdzicie pewnie, że można poradzić sobie z takim błędem dopisując kilka dodatkowych linii programu oraz instrukcje GOTO, które powodują skok do tego nowego fragmentu i z powrotem. Jest to metoda do przyjęcia, jeżeli po tych zabiegach program zacznie działać. Jeśli jednak kiedykolwiek będziecie chcieli powrócić do niego, aby zrobić w nim jakiekolwiek poprawki, napotkacie tylko ciąg poplątanych instrukcji GOTO — jest to program typu spaghetti.

Spójrzcie na treść tego programu:

```
 5 CLS
10 GOTO 100
15 PRINT „ZAGMATWANY"; : GOTO 90
20 PRINT „JEST" : GOTO 70
30 GOTO 50
40 PRINT
50 PRINT „TO"
60 GOTO 20
70 PRINT „MAKSYMALNIE"
80 GOTO 15
90 PRINT „PROGRAM!!"
95 FOR D = 1 TO 2000
96 NEXT D
100 GOTO 30
```

O wiele lepiej jest pisać programy strukturalne, złożone z szeregu oddzielnych, dopasowanych do siebie części. Każdy, kto weźmie taki program do ręki, może go bez trudu przeczytać i zrozumieć. Omówiliśmy już procedury i tablice, możemy więc pisać programy strukturalne jak zawodowi programiści. Procedury są niemal synonimem programowania strukturalnego, ponieważ pozwalają na konstruowanie niezależnych fragmentów, które w dowolnym momencie mogą być wywołane z programu głównego. Gdy korzysta się z procedur, możliwe jest pisanie i testowanie małych fragmentów niezależnie od reszty programu. Procedury stanowią bardzo skuteczną pomoc w budowaniu logicznej struktury programu.

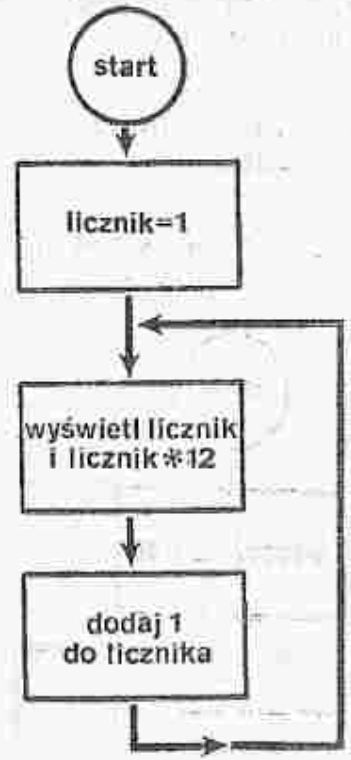
Dysponujemy więc narzędziem programowania niezbędnym do zbudowania dobrego programu; musimy się tylko nauczyć prawidłowego posługiwania się nim. Najważniejszą sprawą jest wstępne przygotowanie — projektowanie programu. Nim przystąpicie do pracy z komputerem, musicie wiedzieć dokładnie, do czego ma służyć wasz program. Musicie skonstruować przejrzysty sposób tzw. przekazywania sterowania w programie. A zatem najlepiej przenieść wszystko na papier, czemu zazwyczaj służy sieć działań. Jest to diagram, który pokazuje główne elementy programu oraz sposób, w jaki przekazywane jest sterowanie między nimi.

Załóżmy, że chcemy napisać program, który wydrukuje tabliczkę mnożenia przez dwanaście. Można w nim wyróżnić cztery zasadnicze części:

1. Początek programu.
2. Część, która przypisuje licznikowi wartość 1.
3. Część, która wyświetla wartość licznika oraz wartość licznika pomnożoną przez dwanaście.
4. Część, która zwiększa wartość licznika o 1.

Z kolei następuje skok w przebiegu programu do początku części 3. Sieć działań tego programu wyglądałaby tak jak na rys. 4.

Każdy program musi mieć punkt startowy. W sieci działań przedstawiamy go zawsze w postaci kółka oznaczonego napisem „start". Linia prowadząca od startu (rys. 4) do kratki znajdującej



Rys. 4. Sieć działań programu mnożenia przez dwanaście

się poniżej pokazuje jedyny możliwy przebieg programu między tymi elementami. Strzałka na tej linii oznacza, że kierunek przebiegu programu musi być od kółka do prostokąta.

Prostokąty znajdujące się w sieci działań reprezentują pewne proste operacje, które należy wykonać. Każdy przedstawia jedną instrukcję Basica albo też niewielką procedurę. Sieć działań pokazuje jedyny możliwy przebieg programu — od pierwszego do drugiego prostokąta, reprezentującego kolejną operację. Linia i strzałka wychodząca z ostatniej kratki pokazują, że program wykonuje skok do tyłu, pomiędzy pierwszy i drugi prostokąt.

Teraz, mając przejrzystą i jednoznaczną sieć działań, możemy z łatwością napisać nasz program. Po prostu do każdego elementu sieci działań podstawiamy jedną lub kilka instrukcji Basica

(z wyjątkiem kółka symbolizującego początek programu, które ma wyłącznie charakter informacyjny):

```
10 LICZNIK = 1
20 PRINT LICZNIK, LICZNIK*12
30 LICZNIK = LICZNIK+1
40 GOTO 20
```

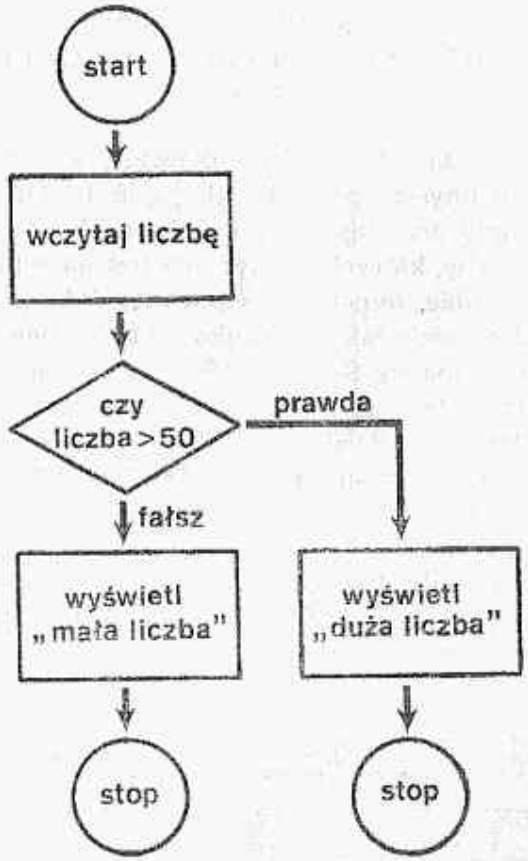
W ten sam sposób możemy napisać sieć działań pierwszego w tym rozdziale, zagmatwanego programu (rys. 5).

Rys. 5. Sieć działań nieprawidłowego programu z poplątanymi instrukcjami

Oto jego treść:

```
10 CLS
20 PRINT „TO JEST MAKSYMALNIE ZAGMATWANY
   PROGRAM"
30 FOR D = 1 TO 2000:NEXT D
40 GOTO 20
```

W większości programów konieczne jest podejmowanie decyzji w trakcie ich wykonywania. Istnieje wtedy więcej niż jeden możliwy przebieg programu. Aby to zaznaczyć, stosujemy w sieci działań specjalny symbol. Rys. 6 przedstawia sieć działań progra-



Rys. 6. Sieć działań zapętlonego programu

mu, w którym trzeba zadecydować, czy liczba jest większa czy mniejsza od 50. Romb oznacza w sieci działań miejsce, w którym trzeba podjąć decyzję. Do rombu prowadzi tylko jedna droga, ale

wychodzą z niego dwie. Wybór jednej z nich zależy od tego, czy zdanie w rombie jest prawdą (true), czy fałszem (false). A skoro mamy dwie niezależne drogi, mamy też dwa możliwe zakończenia programu, które oznaczamy kółkami z napisem „stop".

Znajdującemu się w sieci działań rombowi odpowiada w Basicu instrukcja IF...THEN, a zatem znowu możemy bez trudu przełożyć naszą sieć działań na program napisany w Basicu.

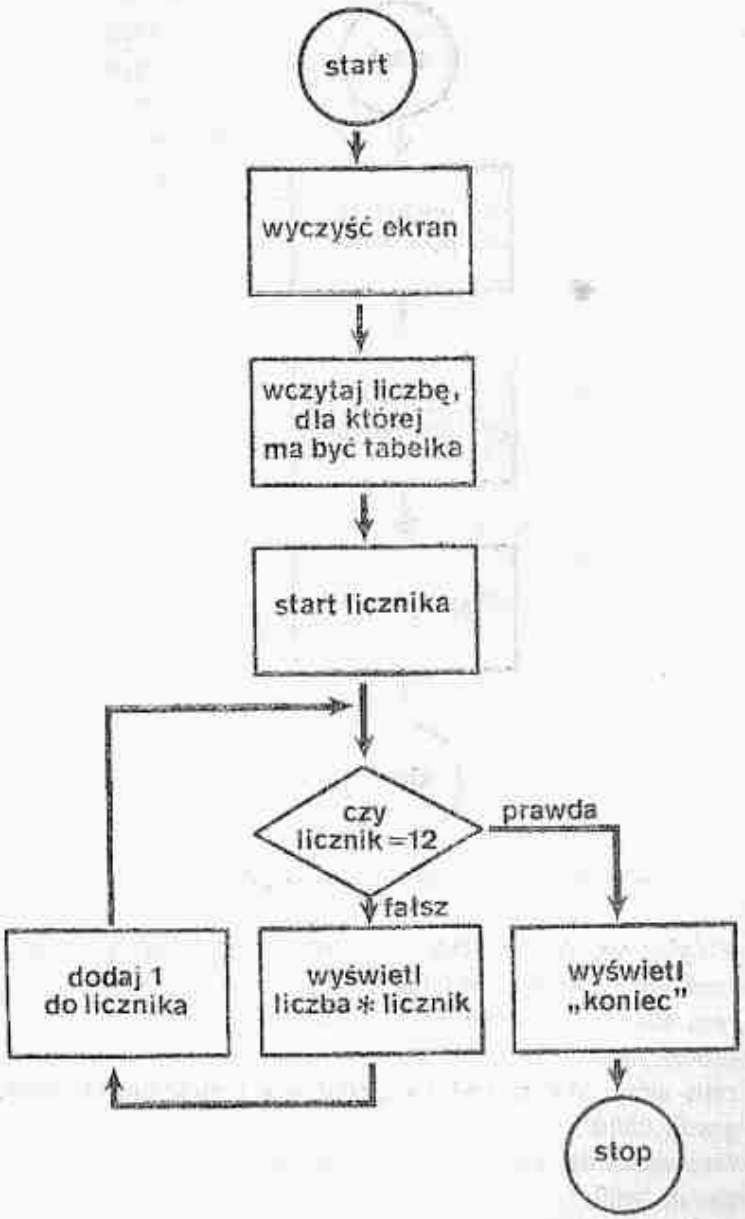
```
10 INPUT „PODAJ LICZBĘ"; LICZBA
20 IF LICZBA > 50 THEN PRINT „DUŻA LICZBA" : END
30 PRINT „MAŁA LICZBA" : END
```

Programy te są tak nieskomplikowane, że nie bardzo wiadomo, po co mielibyśmy pisać dla nich sieć działań. Przydaje się ona wówczas, gdy przystępujemy do pisania bardziej skomplikowanych programów, których budowa nie jest natychmiast widoczna. Wyobraźcie sobie, że potrzebna jest przeróbka programu mnożącego przez dwanaście tak, aby drukował tylko dwanaście pozycji: od jednego do dwunastu. Konstrukcja programu staje się bardziej skomplikowana, zatem uruchomienie go wymaga nieco więcej czasu. Prawidłowa sieć działań pomoże nam upewnić się, że wiemy dokładnie, co nasz program będzie robić, oraz że nie ma w nim miejsca na nieprzewidziane wyniki. Rys. 7 przedstawia sieć działań tego programu.

Z pomocą sieci działań możemy sprawdzić wszystkie możliwe drogi przebiegu programu. Wystarczy potem dokonać podstawienia instrukcji i mamy gotowy program:

```
10 CLS
20 INPUT „PODAJ LICZBĘ, DLA KTÓREJ CHCESZ MIEĆ
     TABELKĘ MNOŻENIA"; LICZBA
30 FOR LICZNIK = 1 TO 12
40 PRINT LICZBA*LICZNIK
50 NEXT LICZNIK
60 PRINT „SKOŃCZONE"
70 END
```

Mając już sieć działań, możecie zbudować szkielet programu stosując instrukcje GOSUB, które odpowiadają elementom sieci



Rys. 7. Konstrukcja bardziej skomplikowanego programu

Rys. 8. Sieć programu ze skokiem do procedury

działań. Gdyby np. sieć działań wyglądała tak jak na rys. 8, moglibyście zbudować taki oto szkielet programu:

```
10 rem skocz do procedury „instrukcje"
20 gosub 1000
30 rem skocz do procedury „kontrola poprawności wejścia"
40 gosub 2000
50 rem skocz do procedury „instrukcje"
60 gosub 3000
70 end
```

Teraz macie już szkielet programu i, aby go skończyć, musicie tylko dopisać procedury.

Zwróćcie uwagę, że sieć działań na rys. 8 zawiera k o n t r o l ę p o p r a w n o ś c i w e j ś c i a. Ten element programu może być bardzo ważny. Jeżeli program nie przewiduje wprowadzenia jakichkolwiek danych przez użytkownika, można mieć pełną kontrolę nad swym dziełem i dokładnie określić, co stanie się w dowolnym miejscu programu. Problemy wyłaniają się w przypadku użycia instrukcji INPUT. Działanie komputera musi opierać się na założeniu, że użytkownik poda sensowne dane. Niestety, nie zawsze tak się dzieje. Zawsze znajdą się ludzie, którzy na proste pytania dadzą nonsensowne odpowiedzi — przypadkiem albo celowo. Jedynym rozwiązaniem jest upewnienie się, że odpowiedź ma sens. Nie daje to pewności, że użytkownik zachowa się tak jak trzeba, ale przynajmniej chroni przed otrzymaniem nonsensownych wyników. Przypuśćmy, że piszecie program, który przetwarza wnioski o wypłatę odszkodowania za uszkodzenie samochodu. Program przewiduje podanie przez użytkownika szacunkowej wartości samochodu. Możecie stwierdzić, że nieprawdopodobne jest, by wartość samochodu była mniejsza niż 50 tysięcy złotych i większa niż 20 milionów złotych. Bardzo łatwo jest wtedy napisać fragment programu sprawdzający, czy podana wartość mieści się w zadanych granicach.

```
100 input wartość
110 if wartość < 50000 or wartość > 20000000 then print
    „spróbuj raz jeszcze"; goto 100
```

Musicie zawsze zakładać, że wasz program zostanie źle użyty bądź też że dostanie się w ręce zupełnych głupców. Dodanie do programu procedur, które pozwalają uniknąć problemów takich jak podany wyżej, zwane jest „uodpornieniem programu na głupotę".

Bez wątpienia słyszeliście o ludziach, od których wymagano zapłacenia rachunku w wysokości 00,00 złotych albo takich, których informowano, że ich czynsz wynosi 50 000 złotych miesięcznie. Mówi się wtedy o „błędach komputerowych", chociaż komputer jest zazwyczaj zupełnie bez winy, czego nie można powiedzieć o programiście. Błąd bowiem pojawia się wtedy, gdy programista nie weźmie pod uwagę wszystkich możliwych dróg przebiegu programu. Może się na przykład zdarzyć fragment programu, w któ-

rym wybrany zostanie jeden wariant, że pewna liczba jest mniejsza od zera, drugi zaś, że jest większa od zera. Zostanie natychmiast pominięty przypadek, gdy liczba ta jest równa zeru.

Jednym ze sposobów uchronienia się przed takimi niespodziankami jest użycie danych testowych. Przygotujcie cały zestaw danych wejściowych tak, aby uwzględnić wszystkie możliwe wartości. Spróbujcie przewidzieć wyniki dla każdego zestawu danych testowych, a następnie porównajcie je z wynikami podanymi przez program. Jeżeli wystąpi niezgodność, będziecie musieli bardzo dokładnie sprawdzić program, notując kolejne zmiany wartości danych testowych w trakcie wykonania programu.

Basic MSX wyposażony jest w dwie instrukcje, które pomagają badać prawidłowość konstrukcji programu. Instrukcja TRON (TRace ON — włączenie śladu) ułatwia śledzenie przebiegu programu. Po wykonaniu tej instrukcji na ekranie wyświetlany będzie numer aktualnie wykonywanej linii. Można więc śledzić działanie programu, aby upewnić się, że jego wykonanie przebiega zgodnie z przewidywaniami. Wykonajcie poniższy prosty przykład, aby zobaczyć, jak działa instrukcja TRON:

```
 10 tron
 20 gosub 100
 30 gosub 200
 40 gosub 300
 50 goto 20
100 return
200 return
300 return
```

Gdy instrukcja pozwalająca na śledzenie programu spełniła już swoje zadanie, można ją wyłączyć posługując się instrukcją TROFF (TRace OFF — wyłączenie śladu).

Sztuka programowania strukturalnego związana jest przede wszystkim z projektowaniem programów. Jeżeli jakiś czas po napisaniu programu będziemy go poprawiać lub przerabiać, musimy zwrócić szczególną uwagę na jego wydruk. Największą rolę odgrywa tu instrukcja REM. Korzystajcie z niej jak najczęściej nie tylko w celu umieszczenia komentarzy w programie, ale i po to, aby nadać mu lepszą szatę graficzną.

Innym sposobem poprawienia czytelności programu jest zrobienie wcięć w pętlach FOR...NEXT, np.:

```
10 rem jest to przykład wcięcia
20:     for i = 1 to 10
30:         for j = 1 to 10
40:             print j*i
50:         next j
60:     next i
70 end
```

W powyższym przykładzie dokonaliśmy wcięcia w każdej linii wchodzącej w skład pętli. Linie, które należą do wewnętrznej pętli, wcięte są bardziej, co zwiększa czytelność. Zwróćcie uwagę, że każda wcięta linia zaczyna się od dwukropka. Jeżeli zostanie on opuszczony, komputer pominie wszystkie spacje znajdujące się na początku linii.

Programowanie strukturalne ma wielu zagorzałych zwolenników, ale są też i tacy, którzy przypisują mu pewne wady. Argumenty przemawiające za i przeciw są — najogólniej mówiąc — następujące:

Zalety

1. Można z łatwością konstruować program i śledzić jego przebieg.
2. Błędy zdarzają się znacznie rzadziej, a w razie wystąpienia znacznie łatwiej je poprawić.
3. Napisanie programu jest prostsze i zajmuje mniej czasu.
4. „Odpluskwianie" (poprawianie) programu zajmuje mniej czasu.
5. Treść programu składa się z oddzielnych procedur, które mogą być wykorzystane w innych programach.

Wady

1. Bywa, że program wykonuje się wolniej.
2. Program zajmuje więcej pamięci.
3. Łatwiej jest usiąść przy klawiaturze i stuknąć w klawisze, niż poświęcić kilka minut na zaplanowanie eleganckiej struktury programu.

11

Operacje tekstowe

W tym rozdziale mamy zamiar przedstawić sposoby operowania tekstami w Basicu MSX. Wiemy już, że Basic umożliwia wykonanie wielu operacji na liczbach, ale to tylko jedna strona medalu. Ostatecznie, gdyby komputer nie umiał robić nic poza prostym przeżuwaniem liczb, wystarczyłby nam zwykły kalkulatorek!

Zdolność wykonywania operacji na tekstach sprawia, że komputer jest czymś innym. Dość często już korzystaliśmy z tekstów, przyjrzyjmy się więc innym sposobom posługiwania się nimi. Operacje tekstowe pozwalają na pisanie znacznie bardziej czytelnych programów. Możemy również pisać programy po prostu dla zabawy.

```
  5 KEYOFF
 10 CLS
 20 PRINT: PRINT „GENERATOR BEŁKOTU"
 30 PRINT: INPUT „ILE SŁÓW CHCESZ UŻYĆ?"; W
 40 DIM W$ (W)
 50 FOR LICZNIK = 1 TO W
 60 PRINT: INPUT „NAPISZ SŁOWO"; W$ (LICZNIK)
 70 NEXT LICZNIK
 80 CLS
 90 PRINT: INPUT „CZY CHCESZ WŁĄCZYĆ GENERA-
    TOR BEŁKOTU? (T/N)" WYBÓR$
100 IF WYBÓR$ = „T" THEN GOSUB 1000 ELSE 140
110 CLS
120 INPUT „JESZCZE RAZ? (T/N)"; WYBÓR$
130 IF WYBÓR = „T" THEN 90
140 CLS: PRINT „DO ZOBACZENIA!!":END
```



```
1000 CLS
1010 FOR E = 1 TO 5
1020 R = INT (RND (1)*W)
1030 PRINT " "; W$ (R);
1040 NEXT E
1050 FOR PAUZA = 1 TO 2000 : NEXT PAUZA
1060 RETURN
```

Program ten jest g e n e r a t o r e m b e ł k o t u. Po prostu łączy on teksty w sposób losowy, generując przedziwną mieszaninę słów. W liniach od 40 do 80 tworzona jest tablica, w której przechowuje się wczytywane słowa. Operacja łączenia tekstu odbywa się w procedurze, która rozpoczyna się w linii 1000. Taki sposób łączenia tekstów wymaga znacznego nakładu pracy. Operacja ta ma bardzo szerokie zastosowanie, potrzebny jest nam więc prosty sposób dokonywania połączeń. Operację tę oznacza się w Basicu przez +, na przykład:

```
10 input „imię"; imię$
20 input „nazwisko"; nazwisko$
30 razem $ = imię $+" "+nazwisko$
40 print „nazywasz się"; razem$
```

Jest to ł ą c z e n i e tekstów. Możemy zbudować w ten sam sposób bardzo złożony tekst.

```
10 a$ = „mo"
20 b$ = „nie"
30 c$ = „te"
40 d$ = „sa"
50 e$ = „wi"
60 print b$+d$+a$+e$+c$
```

Zwróćcie uwagę, że operacja (+) łączenia tekstów i operacja (+) dodawania liczb mniej więcej odpowiadają sobie; jednak żaden inny operator matematyczny nie może być użyty w stosunku do tekstów. Pomimo to możliwe jest wykonywanie szeregu operacji tekstowych z pomocą funkcji tekstowych, w które wyposażony jest Basic MSX.

Funkcja LEFT$ pozwala wyodrębnić pewną liczbę znaków

znajdujących się po lewej stronie tekstu, przez co powstaje zupełnie nowy. LEFT$ potrzebuje dwóch argumentów: jeden wskazuje tekst, na którym ma być wykonana operacja, drugi zaś podaje, ile znaków należy wyodrębnić. Oba argumenty są oddzielone przecinkami, na przykład:

```
10 a$ = „Nieborak"
20 b$ = left$ (a$, 5)
```

Utworzyliśmy nowy tekst, b$, który składa się z pierwszych pięciu znaków tekstu a$. Zwróćcie uwagę, że tekst a$ pozostał nie zmieniony:

```
print a$, b$
```

Ponieważ długość tekstu nie może przekraczać 255 znaków, drugi argument funkcji LEFT$ musi się zawierać w przedziale od 0 do 255. Jeżeli wartość drugiego argumentu jest większa niż liczba znaków w pierwszym argumencie, w wyniku otrzymuje się cały tekst:

```
10 a$ = „cokolwiek"
20 b$ = left$ (a$, 20)
```

Jeżeli drugi argument jest równy zeru, w wyniku otrzymuje się tekst pusty. Może się tak zdarzyć, gdy argumentem tym jest zmienna.

Funkcją dopełniającą do LEFT$ jest RIGHT$. Tworzy ona nowy tekst wyodrębniając znaki z prawej strony tekstu wyjściowego.

```
10 x$ = „tatarak"
20 y$ = right$ (x$, 3)
30 print y$
```

Spośród funkcji operujących na tekstach najbardziej elastyczna jest MID$. Wydobywa ona ciąg znaków z dowolnego miejsca wewnątrz tekstu. Funkcja ta wymaga podania trzech argumentów. Pierwszy oznacza tekst wyjściowy. Drugi oznacza miejsce będące początkiem nowego tekstu. Trzeci argument podaje, ile znaków należy przenieść.

Korzystając z operacji łączenia tekstów zbudowaliśmy słowo „niesamowite"; niech więc teraz MID$ porozbija je znów na części:

```
10 Z$ = „NIESAMOWITE"
20 B$ = MID$ (Z$, 1, 3)
30 D$ = MID$ (Z$, 4, 2)
40 A$ = MID$ (Z$, 6, 2)
50 E$ = MID$ (Z$, 8, 2)
60 C$ = MID$ (Z$, 10, 2)
70 PRINT A$, ,B$, ,C$, ,D$, ,E$
80 END
```

Te trzy funkcje (LEFT$, RIGHT$, MID$) nazywane są funkcjami t n ą c y m i, ponieważ pozwalają przycinać teksty w dowolny sposób. Można korzystać jeszcze z wielu innych funkcji. Prawdopodobnie najbardziej spośród nich przydatna jest funkcja LEN. Podaje ona długość tekstu, czyli liczbę znajdujących się w nim znaków.

```
10 a$ = „Szwecja"
20 print len (a$)
```

Funkcja LEN uwzględnia wszystkie znaki znajdujące się w tekście, a więc spacje, przecinki, kropki itd.

Jednym z najważniejszych zastosowań funkcji LEN jest sterowanie długością pętli tak, by być w zgodzie z długością danego słowa. Przypuśćmy, że chcemy np. wypisać kod każdego znaku w wybranym słowie:

```
10 input w$
20 for i = 1 to len (w$)
30 m$ = mid$ (w$, i, 1)
40 print asc (m$)
50 next i
```

Następny program tworzy anagramy wykorzystując cztery omawiane dotąd funkcje. Jeden z graczy proszony jest o podanie dowolnego słowa tak, aby drugi gracz tego nie widział. Następnie komputer miesza wszystkie litery tego słowa używając w tym celu

procedury w linii 500, po czym pokazuje tę przemieszaną wersję drugiemu graczowi. Ten zaś musi poprzez anagramowanie odnaleźć pierwotny wyraz i podać go komputerowi.

```
 5 KEYOFF
10 CLS
20 INPUT „IMIĘ PIERWSZEGO GRACZA"; G1$
30 INPUT „IMIĘ DRUGIEGO GRACZA"; G2$
40 PRINT G1$; : INPUT „PODAJ SŁOWO DO ANAGRA-
   MOWANIA"; ANA$:CLS
50 GOSUB 500
60 PRINT G2$; : INPUT „JAKIE TO BYŁO SŁOWO?";
   ZGADUJ$
70 IF ANA$ = ZGADUJ$ THEN 80 ELSE 90
80 CLS:PRINT „DOBRZE"; G2$; „TYM SŁOWEM BYŁO";
   ANA$ : GOTO 1000
90 CLS : PRINT „NIEDOBRZE"; G2$; „TYM SŁOWEM
   BYŁO"; ANA" : GOTO 1000
500 L = LEN (ANA$)
510 R = (INT (RND (1)*L))+1
520 M$ = MID$ (ANA$, R, 1)
530 L$ = LEFT$ (ANA$, R−1)
540 R$ = RIGHT$ (ANA$, L−R)
550 PRINT R$, M$, L$
560 RETURN
1000 FOR PAUZA = 1 TO 800: NEXT PAUZA
1010 CLS : INPUT „CZY CHCESZ ZAGRAĆ JESZCZE RAZ?
   (T/N)"; WYBÓR$
1020 IF WYBÓR = „T" THEN 10 ELSE PRINT „DO
   ZOBACZENIA!!" : END
```

Z pomocą funkcji STRING$ można tworzyć teksty składające się z wybranych znaków. Jeżeli chcecie zakończyć pewien fragment programu wyświetleniem równego rządka gwiazdek, zróbcie to w ten sposób:

```
10 for i = 1 to 32
20 print „*";
30 next i
```


Posługując się funkcją STRING$ można to zrobić znacznie łatwiej:

```
10 print string$ (32, „*")
```

Funkcja ta wymaga podania dwóch argumentów. Pierwszy informuje, ile znaków należy wypisać, natomiast drugi — jaki ma to być znak. Można go określić na kilka sposobów. Jeżeli jest to kod ASCII, zostanie wydrukowany znak z takim właśnie kodem, jeżeli zwykły tekst, będzie to pierwszy znak tego tekstu. Aby to sobie lepiej wyobrazić, wykonajcie poniższy przykład:

```
10 print string$ (10, „!")
20 print string$ (12, 65)
30 a$ = „ciastko" : print string$ (14, a$)
40 print string$ (16, „Warszawa")
```

Funkcja STRING$ pozwala nam na eleganckie ulepszanie programów.

Jeżeli chcemy wydrukować rząd spacji, rozwiązanie jest jeszcze prostsze. Korzystamy z funkcji SPACE$. Powoduje ona umieszczenie na ekranie rzędu spacji, na przykład:

```
print space$ (16)
```

i jest prostsza niż pisanie ciągu szesnastu spacji! Argument funkcji musi się zawierać w przedziale od 0 do 255. Jeżeli argument ma część ułamkową, jest ona pomijana.

Funkcja SPC działa podobnie, umieszczając na ekranie puste znaki:

```
print spc (10)
```

Funkcja TAB (tabulator) nie jest dosłownie funkcją tekstową, ale powiemy tu o niej, ponieważ zajmujemy się organizacją wydruku. TAB określa, w którym dokładnie miejscu bieżącej linii ma być wykonana kolejna instrukcja „print". Przypuśćmy, że chcemy napisać liczbę 50 w dziesiątej kolumnie. Powinniśmy napisać:

```
print tab (10); 50
```

Po funkcji TAB musimy umieścić średnik.

Funkcja TAB zakłada, że na ekranie pierwsza z lewej kolumna ma numer zero, a pierwsza z prawej równa jest szerokości ekranu pomniejszonej o jeden.

W jednej instrukcji „print" możemy umieścić dowolną ilość tabulatorów:

```
print 12; tab (8); „kanapa"; tab (16); 45
```

Pamiętajcie, że za każdym razem po funkcji TAB należy napisać średnik. Funkcja TAB jest niezwykle przydatna przy drukowaniu list i tabel, na przykład:

```
10 for i = 1 to 10
20 input „przedmiot"; p$
30 input „cena"; c$
40 print p$; tab (16); c$
50 next i
```

Program pyta o nazwę przedmiotu i o jego cenę. Następnie wypisuje listę przedmiotów wraz z cenami, a funkcja TAB pozwala na elegancką prezentację wyniku.

Jeżeli bieżąca pozycja wydruku znajduje się poza miejscem wskazywanym przez funkcję TAB, funkcja ta nie działa. Na przykład:

```
print tab(20); 10; tab (10); 8
```

Basic MSX wyposażony jest również w inną parę funkcji: VAL i STR$. STR$ zamienia liczbę na postać tekstową. Wyobraźmy sobie, że potrzebna jest nam ilość cyfr pewnej liczby. Nie możemy skorzystać z funkcji LEN, ponieważ operuje ona wyłącznie na tekstach. Musimy więc zamienić wartość liczby na tekst:

```
10 a = 781755
20 a$ = str$(a)
30 print len(a$)
```

Jest tu jednak pewien problem. W wyniku otrzymamy 7, a nie 6. Gdybyście zmienili linię 10, wpisując do niej inną liczbę, znów otrzymalibyście wynik o jeden większy, niż należałoby oczekiwać. Dzieje się tak, ponieważ Basic rezerwuje jedno miejsce na znak (+ lub −), które uwzględnia przy liczeniu długości tekstu. Dzieje się tak zawsze, a zatem przy korzystaniu z funkcji STR$ należy zachować ostrożność. Jeżeli rzeczywiście potrzebna jest wam ilość cyfr liczby bez znaku, musicie zmienić program tak, aby otrzymać wynik pomniejszony o jeden:

```
10 input a
20 a$ = str$ (a)
30 print (len (a$))−1
```

Działanie funkcji VAL jest przeciwne niż STR$ — zamienia tekst na liczbę. Pozwala więc na uniknięcie pewnych ograniczeń związanych z operowaniem tekstami. Tekst „843772" nie może brać udziału w żadnym mnożeniu, dzieleniu ani odejmowaniu. Dodawanie tekstów daje wyniki inne, niż dodawanie liczb, o czym już zresztą była mowa. Funkcja VAL pozwala nam na ominięcie tych trudności:

```
10 a$ = „843772"
20 a = val (a$)
30 b = a*2
40 b$ = str$ (b)
50 print a$
```

Stworzyliśmy więc nowy tekst, który ma wartość dwa razy większą niż wyjściowy tekst a$.

Większość programów wymaga od użytkownika wprowadzania danych: instrukcja INPUT jest więc dla nas bardzo ważna. Przy wprowadzaniu danych z klawiatury w większości przypadków wskazane jest równoczesne wyświetlanie znaku na ekranie. Pozwala to na śledzenie wszelkich ewentualnych błędów i pomyłek, które mogą się pojawić podczas pisania. Dlatego też, gdy instrukcja INPUT czyta dane, na ekranie pojawia się „echo" każdego naciśniętego klawisza. Jednak w pewnych okolicznościach, zwłaszcza podczas korzystania z grafiki, może to być kłopotliwe.

Wyobraźcie sobie grę, w której chodzi o to, by grający nie widział, co napisał jego przeciwnik — tak jest chociażby w omawianej niedawno grze w anagramowanie. Nie należy raczej ocze-

kiwać od przeciwnika, by — gdy macie coś napisać — patrzył w inną stronę. Basic MSX posiada funkcję INPUT$, która radzi sobie z tym problemem. W razie jej użycia na ekranie nie pojawia się echo żadnego znaku napisanego na klawiaturze.

```
10 let a$ = input$ (10)
20 print a$
```

Ten krótki program oczekuje wprowadzenia z klawiatury dziesięciu znaków. Jeżeli zdecydujemy się na wprowadzenie tylko sześciu, powinniśmy na zakończenie nacisnąć klawisz CTRL-c, a następnie RETURN. Zwróćcie uwagę, że INPUT$ jest funkcją, a nie instrukcją. Musi być ona użyta łącznie z innym słowem kluczowym. Nie możemy po prostu napisać:

```
10 input$ (4)
```

— komputer wyświetli w odpowiedzi komunikat o błędzie. Pomyślcie, jak można by zmodyfikować program anagramujący zastępując instrukcję input funkcją INPUT$.

Na zakończenie powiedzmy o funkcji INSTR. Jest ona dosyć ważna, ponieważ pozwala szukać danego tekstu we wnętrzu innego. Możemy więc przeglądać długi tekst, aby sprawdzić pisownię jakiegoś fragmentu albo poszukując pewnej grupy znaków.

```
10 D$ = „zarządzenie"
20 K$ = „rząd"
30 P = INSTR (D$, K$)
40 PRINT „tekst pojawił się na pozycji"; P
```

W trakcie wykonywania powyższego programu komputer będzie przeglądał tekst D$ aż do pierwszego pojawienia się tekstu K$, następnie wyliczy pozycję w D$, na której został znaleziony K$ i przypisze tę wartość zmiennej P.

Jeżeli poszukiwany tekst nie istnieje albo jeżeli oba teksty są puste (nie zawierają żadnego znaku), funkcja INSTR przybiera wartość zero. Jeżeli zaś pusty jest tylko tekst poszukiwany, INSTR równa się jeden.

Możemy zażądać, aby przeglądanie dłuższego tekstu rozpoczęło się od zadanego miejsca.


```
10 DUŻY$ = „laryngolog"
20 MAŁY$ = „gol"
30 S = INSTR (3,DUŻY$, MAŁY$)
40 PRINT „Tekst 'gol' jest na pozycji"; S
```

W powyższym przykładzie poszukiwanie tekstu „gol" rozpoczęło się od trzeciego znaku tekstu „laryngolog".

Funkcje tekstowe mogą oszczędzić wielu powtórzeń i przepisywań; pamiętajcie więc o nich w trakcie pisania programów.

12

Operowanie danymi

Większość programów wymaga od użytkownika wprowadzania informacji w trakcie ich wykonywania, ale prócz tego potrzebne są takie dane, które wykorzystuje się podczas każdego wykonywania programu. Są one zapisane w programie, a zatem można się do nich odwołać w dowolnym momencie. Do ich wprowadzenia możemy użyć szeregu instrukcji podstawienia, ale gdy dotyczy to dużej liczby danych (informacji), jest raczej uciążliwe:

```
10 a = 1
20 b = 29
30 c = 54
40 d = 386
50 e = 738
60 f = 329
70 g = 841
80 h = 314
90 print a; b; c; d; e; f; g; h
```

Uciążliwości te mnożą się w miarę wzrastania liczby danych. Możemy większości ich uniknąć korzystając z instrukcji READ i DATA, których użycie znacznie upraszcza program:

```
10 for i = 1 to 8
20 read a
30 print a
40 next i
50 data 1, 29, 54, 386, 738, 329, 841, 314
```

Gdy komputer napotyka w linii 20 instrukcję READ, zagląda do instrukcji DATA na końcu programu. Pierwszą znalezioną informacją jest liczba 1, a zatem komputer przypisuje zmiennej A wartość 1. W trakcie drugiego obejścia pętli FOR...NEXT zostaje napotkana informacja READ A. Raz jeszcze komputer zagląda do instrukcji DATA, ale tym razem sięga po drugą pozycję na liście danych; jest to liczba 29 — nowa wartość zmiennej A. Proces ten powtarza się, aż do ośmiokrotnego wykonania pętli i wykorzystania wszystkich pozycji na liście instrukcji DATA. READ i DATA muszą zawsze występować razem. Instrukcja DATA zawiera listę danych, które mają być przeczytane instrukcją READ, a każda pozycja na liście musi być oddzielona od sąsiedniej przecinkiem.

W instrukcji DATA mogą znajdować się zarówno liczby, jak i teksty:

```
10 for i = 1 to 5
20 read kolor$
30 print kolor$
40 next i
50 data niebieski, zielony, czerwony, czarny, biały
```

Można również w tym samym programie mieszać ze sobą teksty i liczby:

```
10 read a
20 read x$
30 read b
40 read y$
50 data 78, cena, 383, model
```

Niezwykle ważne jest, aby dane zostały umieszczone we właściwym porządku. Można wprawdzie podstawić liczbę pod zmienną tekstową, ale odwrotnie nie da się zrobić. Zachowajcie też ostrożność korzystając w programie z więcej niż jednej instrukcji READ. Śledzenie programu będzie łatwiejsze, jeżeli wszystkie zmienne zostaną umieszczone w jednej instrukcji READ. W Basicu MSX można to zrobić oddzielając kolejne zmienne przecinkami:

```
10 read w, b$, rozmiar$
```

Skorzystamy z tego w następnym programie, w którym cztery zmienne są czytane za pomocą jednej instrukcji READ:

```
10 REM KONSTRUKCJA OBRAZU
20 CLS
30 PRINT „TEN PROGRAM POKAZUJE UŻYCIE
   INSTRUKCJI READ I DATA"
40 PRINT: INPUT „JESZCZE RAZ? (T/N)"; WYBÓR$
50 REM SZANSA KONTYNUOWANIA
60 IF WYBÓR = „T" THEN 70 ELSE 20
70 CLS
80 REM PONOWNA KONSTRUKCJA OBRAZU
90 PRINT „NR ELEMENTU NAZWA ILOŚĆ CENA"
100 PRINT „.....„
110 REM PRZECZYTAJ PIERWSZY ZESTAW I NAPISZ
   GO
120 FOR PĘTLA = 1 TO 5
130 READ EL, NAZWA$, IL, CE
140 PRINT EL; NAZWA$; IL; CE
150 REM POCZEKAJ CHWILĘ
160 FOR PAUZA = 1 TO 500 : NEXT PAUZA
170 REM POWTÓRZ TO
180 NEXT PĘTLA
190 END
200 REM A TERAZ WSZYSTKIE DANE
210 DATA 100, MNÓSTWO, 10, 100
220 DATA 200, PROSIACZEK, 3.14, 1.83
230 DATA 300, KAPUSTA, 76, 34
240 DATA 400, GWATEMALA, 0, 99
250 DATA 500, PRZEWODNIK, 42, 5
```

Musicie również bardzo uważać, aby instrukcji READ nie zabrakło danych. Jeżeli zostanie ona napotkana po przeczytaniu wszystkich pozycji w instrukcji DATA, ujrzycie komunikat „Out of Data" (poza danymi). Poniżej znajduje się przykład niewłaściwego operowania danymi:

```
10 for i = 1 to 4
20 read liczba
30 next i
40 goto 10
```



```
50 data 21, 36
60 data 44, 32
```

Komputer czyta kolejno pierwsze dwie wartości, po czym przechodzi do czytania następnych dwóch z drugiej instrukcji DATA. Gdy to zrobi, nie może znaleźć kolejnych danych, wobec czego zatrzymuje się sygnalizując błąd. Gdybyśmy chcieli ponownie użyć tych samych danych, musielibyśmy dołączyć do programu instrukcję RESTORE. Jej działanie polega na tym, że gdy komputer ponownie napotka instrukcję READ, odwołuje się do pierwszej w programie pozycji na liście danych. Dopiszcie do programu jeszcze jedną linię:

```
35 restore
```

Hokus pokus! Nasz program działa.

Jeżeli program zawiera więcej instrukcji DATA, za pomocą instrukcji RESTORE możemy dowolną z nich przygotować do ponownego czytania. Na końcu instrukcji RESTORE dodajemy właściwy numer linii:

```
10 R = INT (RND*4)+1
20 S = (R*10)+60
30 READ N, M
40 PRINT N, M
50 RESTORE S
60 GOTO 10
70 DATA 15, 25
80 DATA 23, 48
90 DATA 12, 43
100 DATA 84, 386
```

W liniach 10 i 20 następuje wybranie numeru, który może być równy 70, 80, 90 lub 100. Numery te odpowiadają numerom linii, w których — na końcu programu — znajdują się instrukcje DATA. Linia 50 przygotowuje do czytania dane znajdujące się w linii o wybranym numerze.

Takie użycie instrukcji RESTORE jest bardzo wygodne. Możemy dzięki temu przygotować program zawierający kilka zestawów danych. Instrukcja RESTORE pozwoli wybrać spośród nich

właściwy. Poniżej mamy treść programu-quizu zawierającego trzy zestawy pytań i odpowiedzi. Właściwe dane są wybierane za pomocą instrukcji ON...GOSUB i RESTORE.

```
10 CLS
20 KEYOFF
30 PRINT„      QUIZ MSX"
35 PRINT „WSZYSTKIE ODPOWIEDZI MUSZĄ ZACZYNAĆ SIĘ DUŻĄ LITERĄ"
40 PRINT : PRINT „1. SPORT"
50 PRINT „2. ZWIERZĘTA"
60 PRINT „3. ZIEMIA"
65 S = 0
70 PRINT: INPUT „WYBIERZ 1, 2 LUB 3"; C
80 IR C < 1 OR C > 3 THEN 70
90 ON C GOSUB 1000, 2000, 3000
100 GOTO 10
110 DATA KTO JEST REKORDZISTĄ ŚWIATA W SKOKU W DAL, BOB BEAMON
120 DATA KTO WYGRAŁ NAJWIĘCEJ TURNIEJÓW GOLFOWYCH, JACK NICKLAUS
130 DATA KTO ZDOBYŁ NAJWIĘCEJ MEDALI OLIMPIJSKICH, MARK SPITZ
140 DATA JAKA DYSCYPLINA OLIMPIJSKA ZOSTAŁA WPROWADZONA W 1984 ROKU, PŁYWANIE SYNCHRONICZNE
150 DATA KTO ZDOBYŁ NAJWIĘCEJ BRAMEK W KRYKIECIE, DENNIS LILLIE
200 DATA JAKIE JEST NAJWYŻSZE ZWIERZĘ LĄDOWE, ŻYRAFA
210 DATA JAKIE JEST NAJWIĘKSZE ZWIERZĘ, PŁETWAL BŁĘKITNY
220 DATA JAKI JEST NAJWIĘKSZY BEZKRĘGOWIEC, MĄTWA OLBRZYMIA
230 DATA JAKI SŁOŃ MA NAJWIĘKSZE USZY, AFRYKAŃSKI
240 DATA JAKIE ZWIERZĘ BIEGA NAJSZYBCIEJ, GEPARD
```



```
300 DATA JAKA RZEKA JEST NAJDŁUŻSZA NA ŚWIECIE, NIL
310 DATA JAKA GÓRA JEST NAJWYŻSZA NA ŚWIECIE, MOUNT EVEREST
320 DATA JAKI OCEAN JEST NAJWIĘKSZY NA ŚWIECIE, SPOKOJNY
330 DATA JAKA WYSPA JEST NAJWIĘKSZA NA ŚWIECIE, GRENLANDIA
340 DATA JAKA PUSTYNIA JEST NAJWIĘKSZA NA ŚWIECIE, SAHARA
1000 RESTORE 110
1010 CLS
1020 FOR N = 1 TO 5
1030 READ PYT$, ODP 4
1040 PRINT PYT$: INPUT „ODPOWIEDŹ"; ODPOWIEDZ$
1050 IF WYBÓR$ = ODP$ THEN PRINT „DOBRZE" : S = = S+1: GOTO 1060
1060 PRINT „ŹLE"
1070 NEXT N
1080 PRINT „ZDOBYŁEŚ"; S; „PUNKTÓW NA 5 MOŻLIWYCH"
1090 FOR PAUZA = 1 TO 500 DO : NEXT PAUZA
1100 RETURN
2000 RESTORE 200
2010 GOTO 1010
3000 RESTORE 300
3010 GOTO 1010
```

Używaliśmy już magnetofonu kasetowego do zapisywania programów, aby móc je później wykorzystać. Możemy też zapisywać dane, by wówczas, gdy będzie to wygodne, można się było do nich odwołać. Basic MSX pozwala tworzyć zbiory danych przeznaczone do zapisania na taśmie.

Zasady katalogowania danych w Basicu pod wieloma względami przywodzą na myśl archiwa biurowe. Gdybyście zakładali zwykłą kartotekę do przechowywania określonych informacji, nadalibyście jej nazwę, aby można ją było odróżnić od innych akt archiwalnych. Przed przeczytaniem (albo zapisaniem) jakiejkol-

wiek informacji należy najpierw otworzyć kartotekę (open). Po wykorzystaniu należy ją zamknąć (close), aby nie narobić bałaganu. To samo odnosi się do katalogowania danych w Basicu. Poniżej znajduje się program, który otwiera zbiór o nazwie „dane" i gromadzi w nim pewne informacje:

```
10 OPEN „CAS:DANE" FOR OUTPUT AS # 1
20 READ D$
30 IF D$ = „ZZZ" THEN 90
40 READ D
50 PRINT # 1, D$
60 PRINT # 1, D
70 GOTO 20
80 DATA AB, 1, CD, 2, EF, 3, ZZZ
90 CLOSE # 1
100 END
```

Wyjaśnimy teraz, jak ten program działa. Znajdująca się w linii 10 instrukcja OPEN informuje komputer, że ma on otworzyć zbiór. W słowie „cas" (cassette — kaseta) zawarta jest informacja, że będzie to zbiór kasetowy. Możemy też otwierać zbiory, które prześlą informacje do innych urządzeń:

crt:	cathode ray tube (ekran telewizora)
grp:	graphic screen (ekran graficzny)
lpt:	line printer (drukarka)

Chwilowo zajmiemy się tylko kasetą.

Znajdujące się po dwukropku słowo „dane" informuje komputer, że zbiór będzie miał nazwę „dane". OUTPUT oznacza, że informacja będzie wpisana do zbioru, a nie czytana z niego. AS # 1 określa, że będzie to zbiór o numerze 1; # oznacza po prostu „numer".

Teraz, kiedy powiedzieliśmy komputerowi wszystko, co musi wiedzieć na temat zbioru, możemy przystąpić do zapisywania w nim informacji. Instrukcją, która przesyła dane do magnetofonu kasetowego, jest PRINT # 1. Tak jak normalna instrukcja PRINT przesyła informacje na ekran, PRINT # 1 przesyła informacje do magnetofonu. Linie 20 do 70 odpowiadają za zapisanie wszystkich wartości z instrukcji DATA do zbioru o nazwie „dane", aż

do napotkania tekstu „zzz". Jest to „fałszywa" wartość — wartość, która wskazuje, że napotkany został koniec danych. Gdy tak się stanie, nastąpi skok do linii 90, w której instrukcja CLOSE zamyka # 1.

Po zapisaniu danych na kasecie można zapomnieć o nich aż do chwili, gdy będą potrzebne. Gdy zechcemy sięgnąć do naszych danych, potrzebny będzie inny program, który wprowadzi je ponownie do komputera.

```
10 MAXFILES = 1
20 PRINT „ZAPISANE DANE TO:"
30 OPEN „CAS:DANE" FOR INPUT AS # 1
40 FOR Q = 1 TO 3
50 INPUT # 1, D$
60 INPUT # 1, D
70 PRINT D$, D
80 IF EOF(1) THEN 100
90 NEXT Q
100 CLOSE # 1
110 END
```

Podobnie jak przy zapisywaniu danych, musimy otworzyć zbiór „dane", zanim będziemy mogli uzyskać do niego dostęp; instrukcja ta znajduje się w linii 20. Zwróćcie uwagę, że tym razem zbiór jest otwarty dla wejścia (INPUT), a nie dla wyjścia (OUTPUT), ponieważ teraz wprowadzamy dane z kasety. Istnieje również trzecia możliwość, APPEND (dołączyć) — dopisywanie informacji do istniejącego zbioru.

Jeżeli chcemy wprowadzić dane ze zbioru # 1, używamy instrukcji INPUT # 1. Fragment programu od linii 40 do linii 90 czyta dane z kasety, pozycja po pozycji, przypisując każdą z nich albo zmiennej D$, albo zmiennej D. W linii 80 następuje sprawdzenie, czy nie został napotkany koniec zbioru. EOF (End Of File — koniec zbioru) jest funkcją, która przybiera wartość 1, jeżeli koniec zbioru został napotkany. W przeciwnym razie wartość jej wynosi zero. Liczba umieszczona w nawiasie znajdującym się po EOF jest numerem zbioru, którego dotyczy sprawdzanie. Używamy tej funkcji, aby mieć pewność, że nie czytamy danych znajdujących się poza końcem zbioru. Instrukcja MAXFILES w

linii 10 określa maksymalną liczbę zbiorów, które chcemy mieć równocześnie otwarte.

Użycie magnetofonu kasetowego w komputerze MSX pozwala na dużą elastyczność pracy. Wiemy już, że kod ASCII stanowi standard. Basic MSX pozwala nam zapisywać i ładować programy w kodzie ASCII za pośrednictwem poniższej instrukcji:

save „cas:prognazw"

oraz

load „cas:prognazw"

Znakiem końca zbioru jest kombinacja CTRL Z.

Można również dopisać do pamięci linie programu zapisanego na kasecie w kodzie ASCII. Używamy w tym celu instrukcji MERGE:

merge „cas : prognazw"

Jeżeli dany numer linii pojawi się zarówno w programie znajdującym się w pamięci, jak i w programie zapisanym na kasecie, linia zapisana na kasecie zastępuje znajdującą się w pamięci linię o tym samym numerze.

Pamiętacie zapewne z rozdziału 1, że programy mogą być napisane w kodzie maszynowym. Basic MSX wyposażony jest w dwie instrukcje, które pozwalają zapisać i załadować programy napisane w kodzie maszynowym:

bsave „cas:prognazw", 20000, 22000, 20100

Po instrukcji BSAVE powinny znajdować się trzy parametry. Pierwszy określa początek obszaru pamięci, który ma być zapamiętany, drugi wskazuje jego koniec, a trzeci — miejsce w pamięci, od którego ma się rozpocząć wykonywanie programu. Wypisana powyżej instrukcja spowodowałaby zapisanie całego obszaru pamięci od adresu 20000 do adresu 22000; zawarta jest w niej również informacja, że adres początku programu jest równy 20100. Jeżeli trzeci parametr zostanie pominięty, przyjmuje się, że jest on taki sam jak adres początku.

bload „cas:prognazw", r, 25000

BLOAD ładuje z kasety program zapisany w kodzie maszynowym. Jeżeli został wpisany parametr R, program zostanie uruchomiony bezpośrednio po załadowaniu, startując od adresu wskazanego przez BSAVE.

Jeżeli w BLOAD ostatni parametr jest pominięty, program zostanie załadowany od adresu wskazanego przez BSAVE. Skoro zostanie użyty — każdy adres będzie przesunięty (zwiększony) o tę wartość.

13
Dźwięk

W rozdziale tym mamy zamiar przedstawić muzyczne talenty komputera MSX. Wszystkie komputery MSX mają wbudowany specjalny układ scalony, którego jedynym zadaniem jest generacja dźwięku i muzyki. Układ ten dysponuje dużymi możliwościami, a Basic MSX jest tak skonstruowany, aby ze wszystkich w pełni korzystać. Zetknęliśmy się już z pewnymi dźwiękami, które mogą być generowane przez komputery MSX. W rozdziale 7 wspomnieliśmy, że naciśnięcie klawiszy CTRL G generuje sygnał brzęczyka. Podobny sygnał zawiadamia o popełnieniu błędu.

W pewnych sytuacjach możliwość generowania dźwięków w naszych programach byłaby bardzo wygodna. W Basic można to zrobić przy użyciu instrukcji BEEP.

```
10 ON KEY GOSUB 100
20 KEY(1) ON
30 GOTO 30
100 BEEP
110 RETURN
```

W tym programie instrukcja BEEP pozwala zasygnalizować, że został wciśnięty klawisz funkcyjny. W ten sam sposób możemy powiadomić o zajściu dowolnego, konkretnego wydarzenia w programie. Znacznie bardziej uniwersalną instrukcją jest PLAY (grać), która pozwala programować muzykę w Basicu MSX. Instrukcja PLAY musi być zawsze uzupełniona tekstem, informującym dokładnie komputer, jakie dźwięki mają być zagrane. Gdybyście np. chcieli zagrać dźwięk E, wystarczyłoby napisać:

```
PLAY „E"
```


W ten sam sposób możemy zagrać szereg nut:

```
PLAY „CBAGFEDC"
```

Są to — niemal dokładnie — opadające dźwięki gamy C dur, ale niezupełnie, ponieważ ostatni dźwięk C jest zagrany o jedną oktawę za wysoko. Możemy generować nuty w dowolnej z ośmiu oktaw. Chcemy zagrać sześć dźwięków w trzeciej oktawie, a zatem wstawiamy przed nimi 03.

```
PLAY „C03BAGFEDC"
```

Wszystkie nuty będą od tego momentu wykonywane w trzeciej oktawie, aż do napotkania polecenia zmieniającego oktawę.

By zostać prawdziwymi muzykami, musimy też umieć zagrać nuty obniżone i podwyższone o pół tonu. W tym celu za symbolem dźwięku umieszcza się znak # lub +, by wskazać, że jest on podwyższony o pół tonu, albo też − (minus), co oznacza obniżenie dźwięku o pół tonu. Na przykład:

```
PLAY „07B−AG"
PLAY „02AB03C#DEF+G#A"
```

Możemy też wybierać nuty za pośrednictwem liczb. Każdemu dźwiękowi przypisany jest numer od 1 dla najniższego do 96 dla dźwięku najwyższego. Pauza oznaczona jest przez zero. Warto zapamiętać, że dźwięk C w czwartej oktawie ma numer 36. W tekście muzycznym, czyli tekście znajdującym się za instrukcją PLAY, wstawiamy przed każdą liczbą N:

```
PLAY „N36N38N40N41"
```

Muzyka byłaby dość monotonna, gdybyśmy przez cały czas mieli do czynienia wyłącznie z dźwiękami tej samej długości. Na szczęście, z pomocą parametru L, możemy dokładnie ustalić czas trwania każdej nuty. Jeżeli parametr L został pominięty, Basic zakłada długość L4, a zatem w dotychczasowych przykładach wszystkie dźwięki miały taki właśnie czas trwania. Powinno to dać pewne wyobrażenie o skali parametru L. Im większą wartość mu się przypisze, tym krótszy jest dźwięk. A oto jak długość dźwięku odpowiada standardowej notacji muzycznej:

Długość	Wartość rytmiczna
L1	cała nuta
L2	półnuta
L4	ćwierć nuta
L8	ósemka
L16	szesnastka

i tak dalej, aż dojdziemy do L64 — sześćdziesiątki czwórki. Można wygenerować też triolę i różne inne długości dźwięku. Na przykład L3 jest równoważne jednej trzeciej taktu w rytmie na cztery, a L6 odpowiada trioli ćwierć nut. Spróbujcie wykonać poniższe przykłady:

```
PLAY „05L1CL2DL4EL8FL16GL32AL64B06L1C"
PLAY „04L2A05L6C#EG#L2A"
```

Podobnie jak inne parametry instrukcji PLAY, parametr L obowiązuje aż do ponownego wystąpienia parametru L o innej wartości. Jeżeli chcemy zmienić czas trwania jednego tylko dźwięku, umieszczamy po prostu za jego symbolem wymagany czas trwania. Na przykład:

D#32 jest równoważne L32D#

W miarę jak teksty muzyczne stają się dłuższe i bardziej skomplikowane, coraz trudniej jest rozstrzygnąć, do którego dźwięku należy dany parametr. Wstawiając do tekstu średniki możemy sprawić, że program będzie bardziej przejrzysty. Używamy ich do rozbicia tekstu na mniejsze grupy powiązanych ze sobą znaków. Nie wpływają one na wykonanie muzyki, ale ułatwiają odczytanie treści programu. Poniżej mamy dwie wersje pierwszych kilku taktów hymnu protestanckiego *Angels from the Realms of Glory!* Z którą z nich wolelibyście mieć do czynienia?

```
PLAY „04L4FDB—F05L3DL8C04L4B—F"
```

czy

```
PLAY „04L4F;D;B—;F;05L3D;L8C;04L4B—;F;"
```

Innym sposobem uproszczenia programów muzycznych jest użycie zmiennych tekstowych. Teksty muzyczne niczym się nie

różnią od innych używanych przez nas tekstów, a zatem możemy wykonywać na nich takie same działania.

```
10 A$ = „05D; C#; 04B; A"
20 B$ = „G; F#; ED"
30 PLAY A$+B$
```

Ponieważ wiele modeli składa się z kilku wielokrotnie powtarzających się fraz, możemy znacznie skrócić czas pisania programu. Przyjrzyjcie się, w jaki sposób wykorzystano zmienne tekstowe i łączenie tekstów w tym wykonaniu chorału Bacha *Jesu, meine Freude.*

```
10 CLEAR 400
20 CLS
30 A$ =„04L8G; A; B;"
40 B$ =„05D; C; C; E; D;"
50 C$ =„D; G; F#; G; D; 04B;"
60 D$ =„05C; D; E; D; C;"
70 E$ =„04B; A; B; G;"
80 F$ =„F#; G; A; D; F#; A;"
90 G$ =„05C; 04; B; A; B;"
100 H$ =„04E; 05D; C; 04B; A; G;"
110 I$ =„D; D; F#;"
120 J$ =„G; B; 05D; G; D; 04B;"
130 K$ =„G; B; 05D;"
140 L$ =„05L2G; L4C;"
150 M$ =„L2D; L4D;"
160 N$ =„L2C; 04L4B;"
170 O$ =„05C; 04A; F#; D;"
180 P$ =„F#; A; 05C; 04B; A;"
190 Q$ =„04L8G; B; A; B;"
200 R$ =„F; D; 04B;"
210 S$ =„E; C; 04A;"
220 T$ =„F#; A; 05C; D; 04B; G;"
230 U$ =„E; G; B;"
240 V$ =„E; G; B"
250 W$ =„L2G;"
260 X$ = A$+B$+C$+A$
270 Y$ = D$+E$+F$+G$
```

```
280 Z$ = H$+I$+J$+K$
290 PLAY X$+Y$
300 PLAY X$+Z$
310 PLAY Q$
320 PLAY B$+C$+A$
330 PLAY Y$+X$+Z$
340 PLAY R$+K$
350 PLAY S$+T$+U$
360 PLAY O$+P$+V$
370 PLAY X$+Y$+X$
380 PLAY H$+I$+W$
```

W programie znajduje się instrukcja CLEAR, której dotąd nie napotkaliśmy. Tutaj używamy jej do zarezerwowania dodatkowego obszaru pamięci na nasze zmienne tekstowe. Zwykle na teksty przeznacza się 200 komórek pamięci. Ponieważ nasz program zawiera wiele tekstów, wykorzystana zostaje nie tylko cała ta przestrzeń, ale musimy jeszcze zarezerwować dodatkowy obszar pamięci. Instrukcja CLEAR pozwala określić, ile bajtów (komórek pamięci) przeznacza się na teksty.

Możemy wyznaczyć tempo wykonywania programu za pomocą parametru T. Jeżeli nie podamy wartości tego parametru, która może być dowolną liczbą z przedziału od 32 do 255, komputer zakłada 120. Parametr V wyznacza natężenie dźwięku, z jakim będzie odtwarzana melodia. Przybiera ona wartości od 0 do 15, przyjmując z założenia wartość 8. Powyższe dwa parametry mogą spowodować, że efekty wywołane przez tekst muzyczny w instrukcji PLAY będą bardzo różne:

```
PLAY „T200V6; 02G; B; 03D; T100V12; G; B; 04D; G"
```

Spróbujcie zmieniać natężenie dźwięku i tempo hymnu *Jesu, Joy of Man's Desiring*, aż znajdziecie wartości najlepiej pasujące do tego programu.

Cisza bywa nie mniej ważna niż dźwięki. Parametr R oznacza pauzę. R może przyjmować wartości z przedziału od 1 do 64, a czas trwania pauzy definiowany jest tak samo jak w przypadku parametru L. Poniższy program używa parametru R do wygenerowania sygnału SOS w alfabecie Morse'a:

```
10 A$ = „05L16F; R16; F; R16; F; R8"
20 B$ = „L8F; R8; F; R8; F; R8;"
30 PLAY A$+B$+A$+„R2"
40 GOTO 10
```


Możemy zmienić czas trwania zarówno pauzy, jak i dźwięku, dodając do nuty kropkę. Wykonywany dźwięk będzie wtedy o połowę dłuższy. Inaczej mówiąc, czas trwania dźwięku zostanie pomnożony przez 3/2. Jest to równoważne dodaniu kropki do nuty w normalnej notacji muzycznej.

```
10 PLAY „03L8C.D16E.F16; G2;"
```

Nie wolno umieszczać średnika po nucie opatrzonej kropką, w przeciwnym bowiem razie pojawi się komunikat „Illegal function call" (niedozwolone wywołanie funkcji). Możemy w dalszym ciągu zmieniać czas trwania dźwięku dodając kolejne kropki. Każda z nich ma dwa razy słabsze działanie niż kropka poprzedzająca, powodując o połowę mniejsze wydłużenie czasu trwania dźwięku. Użyjemy kropkowanych nut, aby dokonać jeszcze jednej zmiany w chorale *Jesu, meine Freude*. Przyjrzyjmy się linii 40 programu:

```
40 B$ = „05D; C; C; E; D;"
```

Zwróćcie uwagę, że dźwięk C powtarza się kolejno dwa razy. Stwarza to pewien problem ze względu na skuteczność działania generatora dźwięku. Komputer łączy kolejne dźwięki tak gładko, że nie jesteśmy w stanie rozróżnić końca jednego dźwięku i początku następnego. Nie jest to dokładnie ten efekt, o który chodziło Bachowi, i dlatego wstawiamy pomiędzy te dwa dźwięki bardzo krótką pauzę, aby podkreślić początek drugiego dźwięku. Kropka pozwoli nam ustawić długość pierwszego dźwięku tak, aby dopasować się do pauzy.

```
40 B$ = „05D; C8..R64; C4E; D;"
```

Program nasz będzie bardziej elastyczny, jeżeli do tekstów muzycznych wprowadzimy zmienne. W tym celu musimy umieścić przed zmienną znak =, aby poinformować komputer, że następna wartość jest zmienną:

```
10 A = INT (RND(1)*150)+50
20 PLAY „T = A; C; D; E; F; G;"
30 GOTO 10
```

Zmienne mogą być szczególnie użyteczne, jeżeli połączy się je z parametrem N.

```
10 FOR I = 1 TO 96
20 PLAY „N = I;"
30 NEXT I
```

Najbardziej jednak uniwersalnym sposobem korzystania ze zmiennych jest włączenie zmiennych tekstowych do tekstów muzycznych. Wstawiamy wtedy X bezpośrednio przed daną zmienną:

```
10 A$ = „E; G#; B"
20 B$ = „C#; E; A"
30 C$ = „D#; F#; B"
40 D$ = „A; G#; F#; E2"
50 P$ = „XA$; XB$;XC$; XD$;"
200 PLAY P$
```

Gdy korzysta się ze zmiennych dowolnych typów, bardzo ważne jest dołączenie do nich średników. Ostatniej linii przypisaliśmy duży numer, ponieważ mamy zamiar rozszerzyć nieco nasz program. Komputer może generować równocześnie trzy różne melodie i dlatego wykonywana przezeń muzyka jest tak fascynująca. Układ scalony w komputerze MSX ma trzy niezależne kanały muzyczne, które można równocześnie uruchamiać. Wystarczy określić, które teksty muzyczne mają być równocześnie wykonywane, włączając je — oddzielone przecinkami — do tej samej instrukcji PLAY. Wprowadźcie poniższą instrukcję w trybie konwersacyjnym, aby nie zaburzać naszego programu:

```
PLAY „A", „C#", „E"
```

Usłyszeliśmy akord A dur! Komputer wykonał trzy teksty równocześnie. Wykorzystamy tę właściwość, aby włączyć do naszego małego utworu dodatkową linię melodyczną. Dopiszcie do programu poniższe linie:

```
60 E$ = „05E2; F#8; D#8"
70 F$ = „C#2; E8; D#8"
80 G$ = „04B; 05C#; D#; F#; E1"
90 Q$ = „XE$; XF$; XG$;"
```


oraz zmieńcie linię 200 na:

```
200 PLAY P$, Q$
```

Daje to pełną „dwuczęściową" melodię. Do wykończenia jej potrzebna nam będzie jeszcze linia basowa w trzecim kanale:

```
100 H$ = „03E2.A2.B2.E; G#; B; E2"
200 PLAY P$, Q$, H$
```

Możemy wreszcie ująć całość w instrukcję pętli tak, aby dwukrotnie wykonać cały utwór.

```
5 FOR A = 1 TO 2
210 NEXT A
```

Jak już widzieliśmy, wykorzystywany przez komputery MSX układ scalony, który służy do generacji dźwięków, ma ogromne możliwości. Basic MSX pozwala wykorzystać je w pełni za pomocą instrukcji SOUND (dźwięk). Generator dźwięków zawiera szereg rejestrów, które definiują dźwięk wytwarzany przy użyciu instrukcji PLAY.

Działanie rejestrów

REJESTR	DZIAŁANIE
0	Dokładne strojenie częstości dźwięku w kanale A
1	Zgrubne strojenie częstości dźwięku w kanale A
2	Dokładne strojenie częstości dźwięku w kanale B
3	Zgrubne strojenie częstości dźwięku w kanale B
4	Dokładne strojenie częstości dźwięku w kanale C
5	Zgrubne strojenie częstości dźwięku w kanale C
6	Częstość szumu
7	Włącz
8	Amplituda w kanale A
9	Amplituda w kanale B
10	Amplituda w kanale C
11	Dokładne strojenie częstości obwiedni
12	Zgrubne strojenie częstości obwiedni
13	Kształt obwiedni

Działanie każdego rejestru najlepiej poznać przez doświadczenie. Umożliwia ono znacznie lepsze zrozumienie działania in-

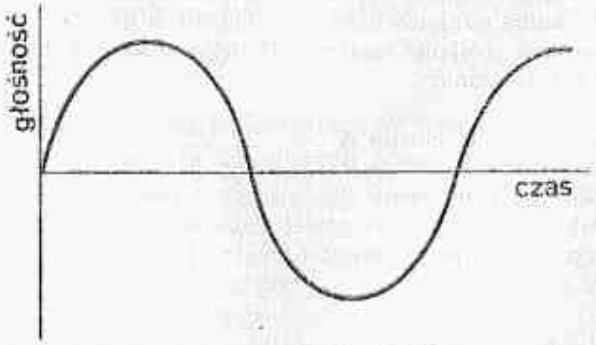
strukcji SOUND niż jakiekolwiek wyjaśnienia techniczne. Instrukcja SOUND ma następującą składnię:

SOUND 8,255

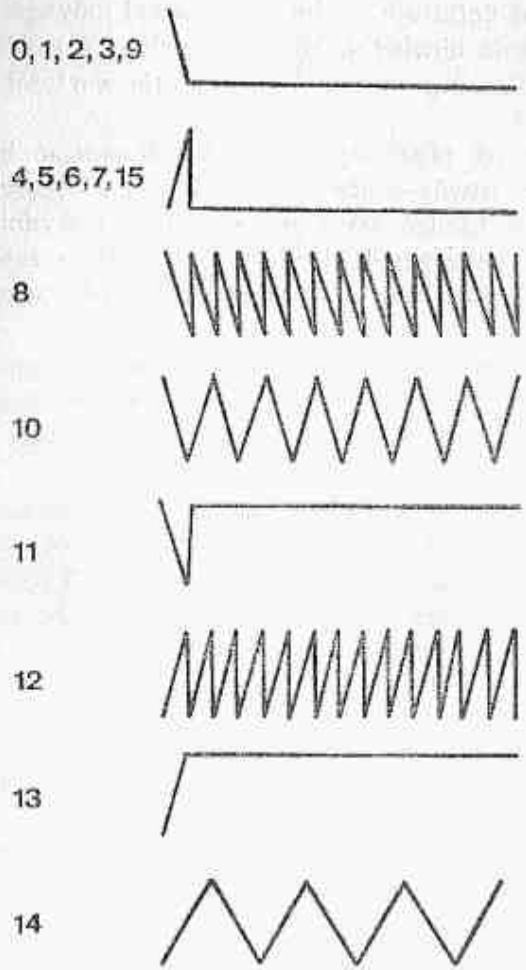
W przykładzie tym rejestrowi 8 przypisana została wartość 255. Jeżeli spróbujecie wprowadzić wartość większą niż 255, zostanie zasygnalizowany błąd. „Illegal function call". Wartości, jakie przypiszecie każdemu z rejestrów, wpłyną na końcowy efekt dźwiękowy. Pewną pomocą może być poniższy program, który generuje nieco interesujących dźwięków:

```
10 INPUT „KSZTAŁT(0—15)"; KSZTAŁT
20 INPUT „MODULACJA(1—65535)"; MDL
30 FORT I = 1 TO 7
40 SOUND I,0
50 NEXT I
60 SOUND 10,17
70 SOUND 11,19
80 SOUND 12,23
90 SOUND 13,4
100 SOUND 9,7
110 PLAY „T64; S = KSZTAŁT; M = MDL; A"
120 GOTO 110
```

Program ten nie tylko jest źródłem pewnych niezwykłych efektów dźwiękowych, ale wprowadza również dwa nowe parametry: S i M. Są one używane w połączeniu z instrukcją SOUND



Rys. 9. Obwiednia tonu generowanego przez brzęczyk komputera MSX



Rys. 10. Różne wartości parametru S

i sterują rodzajem dźwięku generowanego w instrukcji PLAY. Decydują one o kształcie obwiedni dźwięku. Jest to graficzne przedstawienie fali dźwiękowej, ukazujące zmiany natężenia dźwięku w funkcji czasu. Na rys. 9 mamy obwiednię tonu czystego, takiego, jaki jest generowany przez brzęczyk komputera MSX. Rys. 10 pokazuje kształty odpowiadające różnym wartościom S. Jeżeli nie jest podana żadna wartość, przyjmuje się z założenia wartość S równą 8.

Parametr M definiuje modulację. Inaczej mówiąc, wpływa on na czas wykonania obwiedni. Można ją ustalić z bardzo dużą dokładnością, ponieważ parametr M przyjmuje wartości z przedziału od 1 do 32767.

Niezależnie od tego, czy chcielibyście napisać koncert, czy tylko wzbogacić utwór o efekty dźwiękowe, zdolności muzyczne komputera MSX bardzo wam pomogą. Można wydobyć z niego ogromne bogactwo różnych dźwięków. Znajdźcie zatem czas na doświadczenia — będziecie zaskoczeni swoimi odkryciami.

14
Kolor i grafika

Najbardziej chyba fascynującą właściwością komputerów domowych jest zdolność operowania kolorem i rysunkiem. W następnych trzech rozdziałach omówimy możliwości graficzne komputerów MSX.

Zacznijmy od koloru. Komputery MSX są w stanie wygenerować szesnaście różnych barw, które można wybierać za pośrednictwem instrukcji COLOR. Przyjrzyjcie się obrazowi, jaki znajduje się na ekranie po włączeniu komputera. Tekst jest biały, tło ciemnoniebieskie, a brzegi kobaltowe. Bardzo łatwo możemy zmienić te barwy:

```
color 10,1,4
```

(Uważajcie, musicie napisać „color", a nie „colour".)

Każdemu z szesnastu różnych kolorów odpowiada liczba, którą umieszcza się w instrukcji COLOR, wyznaczając barwę tekstu (atramentu), tła i brzegów. Liczby związane z kolorami mają następujące znaczenie:

0	przezroczysty
1	czarny
2	zielony
3	jasnozielony
4	ciemnoniebieski
5	jasnoniebieski
6	ciemnoczerwony
7	kobaltowy
8	czerwony
9	jasnoczerwony
10	ciemnożółty

11 jasnożółty
12 ciemnozielony
13 magenta (szkarłatny)
14 szary
15 biały

Poniższy program prezentuje pełną gamę barw komputera MSX:

```
10 FOR A = 1 TO 15
20 COLOR A,1,1
30 GOSUB 150
40 NEXT A
50 FOR B = 1 TO 15
60 COLOR 1,B,1
70 GOSUB 150
80 NEXT B
90 FOR C = 1 TO 15
100 COLOR 15,1,C
110 GOSUB 150
120 NEXT C
130 COLOR 15,4,7
140 END
150 FOR D = 1 TO 400: NEXT D
160 RETURN
```

Możemy również zmieniać format obrazu na ekranie, używając w tym celu instrukcji SCREEN. Możliwe są cztery formaty:

SCREEN 0 mod tekstowy 40*24
SCREEN 1 mod tekstowy 32*24
SCREEN 2 mod o wysokiej rozdzielczości
SCREEN 3 mod wielobarwny

Na przykład wprowadzając w modzie (trybie) konwersacyjnym

```
SCREEN 1
```

otrzymamy format, który daje 24 wiersze po 32 znaki w wierszu. Jeżeli zaś napiszemy:

```
SCREEN 0
```


będziemy mieli na ekranie 24 wiersze po 40 znaków każdy. Za chwilę omówimy pozostałe dwa tryby. Instrukcja SCREEN może być rozbudowana o szereg dodatkowych (do pięciu) parametrów, na przykład:

```
SCREEN 3,2,0,2,0
```

Używaliśmy dotąd pierwszego parametru, który określa format ekranu. Drugi pozwala zdefiniować format tak zwanych skrzatów. Omówimy je w rozdziale 16. Podczas normalnej pracy komputer przy każdym naciśnięciu klawisza wydaje z siebie cichy trzask, ale trzeci parametr instrukcji SCREEN pozwala na włączanie i wyłączanie tego dźwięku. Jeżeli parametr ten jest równy zeru — dźwięk nie jest emitowany, dowolna inna jego wartość powoduje włączenie go. Komputery MSX mogą zapisywać programy na taśmie z prędkością 1200 albo 2400 bodów (bod jest miarą prędkości zapisu lub ładowania programów). Jeżeli czwarty parametr wynosi 1, program zapisywany jest z prędkością 1200 bodów, jeżeli 2 — z prędkością 2400 bodów. Piąty parametr wskazuje, czy dana drukarka jest zgodna (kompatybilna) z komputerami MSX, czy też nie. Zero oznacza brak tej zgodności.

Z naszego punktu widzenia najważniejsze są dwa pierwsze parametry, ponieważ dotyczą one właściwości graficznych. Korzystanie z programów graficznych oraz pisanie gier należy do najbardziej rozrywkowych cech komputerów. Basic MSX, dzięki dużym możliwościom graficznym, umożliwia pisanie pewnych interesujących programów graficznych. Spróbujcie poćwiczyć użycie instrukcji PRINT w połączeniu z pewnymi specjalnymi znakami, którymi dysponują komputery MSX. Możecie stworzyć w ten sposób fascynujące obrazy. Aby uprościć ten rodzaj programowania, musimy nauczyć się, jak umieszczać kursor w dowolnym miejscu ekranu. Najlepiej nadaje się do tego instrukcja LOCATE (umieszczać). Na przykład:

```
10 locate 14,6
20 print „A"
```

Litera A pojawi się na ekranie w czternastej kolumnie szóstego rzędu.

W wielu programach, zwłaszcza w grach, bardzo istotna jest

możliwość przesuwania obiektów po ekranie. Czyni to właśnie poniższy program, który korzysta z instrukcji LOCATE, a ruch jest kierowany za pośrednictwem klawiszy sterujących kursorem.

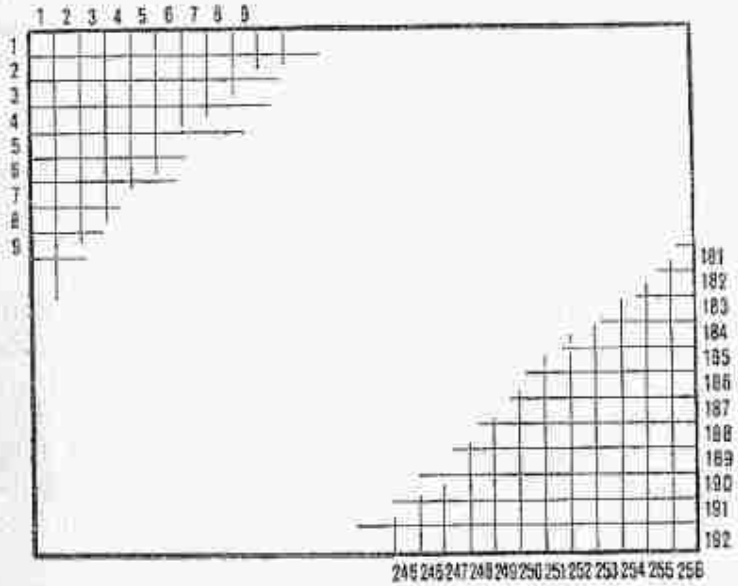
```
10 CLS
20 KEYOFF
30 SCREEN 0
40 PRINT „UŻYWAJ KLAWISZY STERUJĄCYCH
   KURSOREM DO PRZESUWANIA ZNAKU"; CHR$ (129);
   „PO EKRANIE"
50 X = 20 : Y = 12
60 LOCATE X, Y
70 PRINT CHR$ (129)
80 FOR D = 1 TO 50 : NEXT D
90 LOCATE X, Y
100 PRINT CHR$ (32)
110 A$ = INKEY$
120 IF A$ = CHS$ (28) THEN X = X+1
130 IF A$ = CHR$ (29) THEN X = X−1
140 IF A$ = CHR$ (30) THEN Y = Y−1
150 IF A$ = CHR$ (31) THEN Y = Y+1
160 GOTO 60
```

Program ten wprowadza nowe ważne słowo kluczowe — INKEY$. Komputer, gdy napotka to słowo w linii 110, sprawdza natychmiast, czy nie został naciśnięty jakiś klawisz. Jeżeli została np. naciśnięta litera H, zostanie ona podstawiona pod zmienną A$. Jeżeli nie został naciśnięty żaden klawisz, A$ będzie tekstem pustym. Instrukcja ta umożliwia pisanie programów, w których naciśnięcie odpowiedniego klawisza daje natychmiastową reakcję.

Linie 120 do 150 określają kierunek, w którym będzie się poruszał nasz obiekt. Kody 28, 29, 30 i 31 odpowiadają klawiszom oznaczonym strzałkami skierowanymi odpowiednio w prawo, w lewo, w górę i w dół. Jeżeli np. naciśniemy klawisz ze strzałką skierowaną w lewo, instrukcja INKEY$ spowoduje podstawienie CHR$ (29) pod zmienną A$. Zatem zmienna X w linii 120 zostanie zmniejszona o jeden. Ponieważ zaś X określa położenie obiektu na ekranie (linia 60), znak przesunie się o jedno miejsce w lewo.

Jednakże w trybie tekstowym nasze możliwości są częściowo

ograniczone. Aby wykorzystać instrukcje graficzne, musimy przejść do modu drugiego lub trzeciego (SCREEN 2 lub SCREEN 3). W obu tych przypadkach ekranowi zostanie przypisana siatka o 256 kolumnach i 192 rzędach (rys. 11).



Rys. 11. Siatka na ekranie komputera umożliwiająca programowanie graficzne

Każdy element tej siatki nazywany jest punktem obrazowym (pixel). Programowanie graficzne polega na zabarwieniu tego punktu wybranym kolorem. Różnica między trybami SCREEN 2 i SCREEN 3 polega na sposobie korzystania z tych punktów. Wyjaśnimy to wprowadzając instrukcję PSET.

```
10 SCREEN 2
20 PSET (128,96), 1
30 GOTO 30
```

W linii 10 następuje przestawienie ekranu komputera do trybu 2. Linia 30 służy utrzymywaniu komputera w tym modzie. Gdy następuje zakończenie pracy programu, komputer automatycznie powraca do trybu tekstowego, a zatem potrzebna jest nam

linia 30, dzięki której uniknięmy nagłego wyświetlenia komunikatu O.K. Instrukcja PSET została tu użyta w celu nadania punktowi obrazowemu żądanej barwy. Dwa parametry ujęte w nawiasy odnoszą się do współrzędnych siatki z rys. 11. Wskazują one poziomą i pionową pozycję punktu obrazowego, który ma zostać zabarwiony. Są to współrzędne używane przez instrukcję PSET. Trzeci parametr określa kolor, który ma być przypisany punktowi obrazowemu.

Wykonanie tego programu spowoduje zabarwienie na czerwono punktu obrazowego znajdującego się w środku ekranu. Zmieńcie teraz linię 10:

```
10 SCREEN 3
```

i ponownie wykonajcie program. Nastąpi istotna zmiana. Zamiast jednego czerwonego punktu otrzymacie wypełniony czerwienią kwadrat o boku równym czterem punktom obrazowym. Instrukcje graficzne w trybie 2 dotyczą zawsze pojedynczego punktu, podczas gdy w trybie 3 odnoszą się do kwadratu o boku długości czterech punktów obrazowych. Współrzędne, będące parametrami instrukcji PSET, wskazują w modzie 3 położenie punktu środkowego.

Mod 2 pozwala na wykonanie operacji graficznych, w których będą widoczne drobne szczegóły. Mod 3 nadaje się bardziej do prostej, wielobarwnej grafiki. Poniżej mamy program — szkicownik, który pozwoli wykorzystać możliwości graficzne komputera MSX. Używa on klawiszy sterujących ruchem kursora po ekranie i rysowania z jego pomocą wybranym kolorem:

```
10 CLS
20 PRINT „SZKICOWNIK"
30 PRINT : INPUT „CZY POTRZEBUJESZ
   PRZYPOMNIENIA (T/N)"; W$
40 IF W$ = „T" THEN 50 ELSE 100
50 CLS: PRINT „SZKICOWNIK"
60 PRINT „WYBIERZ NUMER KOLORU POMIĘDZY
   1 I 15"
70 PRINT: PRINT „RYSUJ Z POMOCĄ KLAWISZY
   STERUJĄCYCH KURSOREM"
```



```
80 INPUT „JESZCZE RAZ (T/N)"; Z$
90 IF Z$ = „T" THEN 50 ELSE 100
100 INPUT „WYBIERZ KOLOR (1 DO 15)"; K
110 IF C<1 OR C>15 THEN 100 ELSE 120
120 X = 128 : Y = 96
130 SCREEN 3
140 PSET (X, Y), K
150 I$ = INKEY$
160 IF I$ = CHR$ (28) THEN X = X+2
170 IF I$ =CHR$ (29) THEN X = X−2
180 IF I$ = CHR$(30) THEN Y = Y−2
190 IF I$ = CHR$ (31) THEN Y = Y+2
200 GOTO 140
```

Program ten nie tylko ukazuje możliwości graficzne komputera MSX, ale pozwala również na tworzenie różnych ciekawych wzorów. Każdy jednak artysta popełnia od czasu do czasu błędy, możliwość ich poprawiania byłaby więc rzeczą cenną. Służy temu instrukcja będąca dopełnieniem instrukcji PSET. Jest to instrukcja PRESET:

```
10 SCREEN 3
20 PSET (100,100), 6
30 PRESET (100,100)
40 GOTO 40
```

Po uruchomieniu tego programu ujrzycie pusty ekran. Wprawdzie instrukcja PSET w linii 20 spowodowała wyświetlenie czarnego kwadratu na ekranie, ale kwadrat ten został natychmiast skasowany i wypełniony kolorem tła, a sprawiła to instrukcja PRESET w linii 30. Wiemy już zatem dokładnie, jak działa ta instrukcja, przywraca mianowicie kwadratowi o podanych współrzędnych aktualny kolor tła. Możliwe jest rozszerzenie instrukcji PRESET tak, aby określić, którym kolorem chcielibyśmy wypełnić wskazany kwadrat, ale wtedy działałaby ona dokładnie tak samo jak instrukcja PSET.

A zatem rozbudujmy teraz nasz program tak, abyśmy mogli rysować i zmazywać:

```
10 SCREEN 0
```

```
 20 KEYOFF
 30 CLS
 40 PRINT „SZKICOWNIK"
 50 PRINT : INPUT „CZY POTRZEBUJESZ PRZYPOMNIENIA (T/N)"; W$
 60 IF W$ = „T" THEN 70 ELSE 130
 70 CLS : PRINT „SZKICOWNIK"
 80 PRINT : PRINT „WYBIERZ NUMER KOLORU MIĘDZY 1 I 15"
 90 PRINT : PRINT „RYSUJ ZA POŚREDNICTWEM KLAWISZY STERUJĄCYCH KURSOREM"
100 PRINT : PRINT „ABY ZMIENIĆ KOLOR NACIŚNIJ SPACJĘ, ABY ZMAZAĆ NACIŚNIJ 0, UŻYWAJ KLAWISZY STERUJĄCYCH KURSOREM"
110 INPUT „JESZCZE RAZ"; P$
120 IF P$ = „T" THEN 70 ELSE 130
130 INPUT „WYBIERZ KOLOR (1 DO 15)"; K
140 IF K<1 OR K>15 THEN 130 ELSE 150
150 X = 128 : Y = 96
155 COLOR K, 1,1
160 SCREEN 3
170 PSET (X, Y), C
180 GOSUB 1000
190 GOTO 170
1000 KURSOR$ = INKEY$
1010 IF KURSOR$ = CHR$ (28) THEN X = X+2
1020 IF KURSOR$ = CHR$ (29) THEN X = X−2
1030 IF KURSOR$ = CHR$ (30) THEN Y = Y−2
1040 IR KURSOR$ = CHR$ (31) THEN Y = Y+2
1050 IF KURSOR$ = CHR$ (32) THEN GOSUB 2000
1060 IF KURSOR$ = CHR$ (48) THEN GOSUB 3000
1070 RETURN
2000 COLOR 15,4,7
2010 PRINT : INPUT „CZY CHCESZ ZMIENIĆ KOLOR (T/N)"; Z$
2020 IF Z$ = „T" THEN 2040
2030 RETURN 150
2040 PRINT : INPUT „WYBIERZ KOLOR"; K
```



```
2050 IF K<1 OR K>15 THEN 2040
2060 RETURN 155
3000 X = 128 : Y = 96
3010 GOSUB 1000
3020 PRESET (X, Y)
3030 GOTO 3010
```

Istnieje też funkcja, której możemy użyć w każdym z modów graficznych, aby odczytać kolor dowolnego punktu obrazowego. Funkcja POINT (punkt) przybiera wartość równą numerowi koloru punktu o wskazanych współrzędnych. Przydaje się ona bardzo w grach i podobnych im programach:

```
10 SCREEN 2
20 FOR I = 1 TO 10
30 PSET (55+I, 70), 7
40 FOR J = 1 TO 100: NEXT J
50 IF POINT (60,70) = 7 THEN BEEP
60 NEXT I
```

15

Nowe horyzonty

Używając instrukcji PSET jesteśmy wprawdzie w stanie wykonać dowolny rysunek, ale korzystanie z niej — zwłaszcza w bardziej skomplikowanych pracach — może być nieporęczne. Gdybyśmy chcieli wykonać w trybie o wysokiej rozdzielczości rysunek, który wypełnia cały ekran, oddzielne rysowanie każdego punktu byłoby nudne. Basic MSX posiada pewne bardzo poręczne instrukcje graficzne, dzięki którym grafika komputerowa jest znacznie łatwiejsza. Przedstawimy je w tym rozdziale wykonując rysunek domu.

Musimy przede wszystkim narysować kontur domu. Moglibyśmy rysować linie proste z pomocą umieszczonej w pętli FOR...NEXT instrukcji PSET, ale o wiele łatwiej jest skorzystać z instrukcji LINE (linia), która pozwala nakreślić na ekranie linię prostą między dowolnymi dwoma punktami:

```
10 SCREEN 2
20 LINE (10,10)-(240,180)
30 GOTO 30
```

Instrukcja w linii 20 powoduje narysowanie prostej w ustalonej wcześniej barwie (kolorze atramentu) między punktami o współrzędnych (10,10) i (240,180).

Gdybyśmy chcieli rysować innym kolorem, możemy rozszerzyć instrukcję LINE. Zmieńcie linię 20 na:

```
20 LINE (10,10)-(240,180), 6
```

Nowa linia będzie miała kolor 6 — ciemnoczerwony. Jeżeli dodamy następny parametr — B, zamiast linii prostej zostanie narysowany prostokąt, którego dwa przeciwległe wierzchołki będą miały wskazane współrzędne:


```
20 LINE (10,10)-(240,180), 6,B
```

Gdybyście chcieli narysować prostokąt w kolorze atramentu, możecie pominąć parametr określający kolor, zachowując jednak odpowiadający mu przecinek:

```
20 LINE (10,10)-(240-180), ,B
```

Zasada ta odnosi się do większości instrukcji Basica MSX, których parametry oddzielone są przecinkami. Jeżeli zastąpimy B przez BF, prostokąt zostanie zamalowany wybranym kolorem:

```
20 LINE (10,10)-(240,180), 6,BF
```

Możemy przystąpić teraz do wykonania obrazka, rysując kontur domu i ogrodu przy użyciu instrukcji LINE.

```
10 COLOR 15,4,4
20 SCREEN 2
30 CLS
40 LINE (0,90)-(255,192), 3,BF
50 LINE (50,50)-((170,130), 15,BF
60 LINE (50,50)-(110,20),8
70 LINE (110,20)-(170,50),8
80 LINE (50,50)-(170,50),8
250 GOTO 250
```

Nie możemy, niestety, użyć instrukcji LINE do pomalowania dachu. Możemy zamiast tego skorzystać z instrukcji PAINT (malować), która zamalowuje dowolny kształt.

```
90 PAINT (60,48),8,8
```

Współrzędne (60,48) wskazują punkt, w którym ma się rozpocząć malowanie. Następny parametr definiuje kolor farby, a ostatni podaje kolor, który ma być uważany za granicę obszaru. Obszar otoczony tą granicą zostanie całkowicie zamalowany. Upewnijcie się, że granica nie ma żadnych „dziur", w przeciwnym bowiem razie farba „wycieknie" i zaleje cały ekran. W trybie o wysokiej rozdzielczości (mod 2) kolor granicy musi być taki sam, jak kolor farby. W trybie wielobarwnym (mod 3) mogą być one różne.

Nasz dom potrzebuje drzwi wejściowych:

```
100 LINE(105,110)-(115,129),7,BF
```

oraz światła dziennego:

```
110 CIRCLE (220,30), 13,11
120 PAINT (220,30),11
```

Fakt, że instrukcja CIRCLE (okrąg) rysuje okręgi, nie powinien was zaskoczyć. Liczby w nawiasach są współrzędnymi środka okręgu, następny parametr podaje promień (odległość od środka do brzegu), a ostatni określa kolor, którym okrąg ma być rysowany. Instrukcja PAINT pozwoli zamalować świetlik na żółto.

Jeżeli chcemy narysować tylko fragment okręgu, musimy dodać do instrukcji CIRCLE jeszcze dwa parametry. Pierwszy z nich określa punkt startowy, a drugi — końcowy. Mogą one przybierać wartości z przedziału od 0 do 2π (w przybliżeniu 6,28). Możemy wreszcie dodać jeszcze jeden parametr, który zmienia okrąg w elipsę. Jest to ułamek określający stosunek wysokości elipsy do jej szerokości. Jeżeli, na przykład, napiszemy 1/4, zostanie narysowana elipsa, której szerokość będzie cztery razy większa od wysokości. Wykorzystamy ten parametr, aby na naszym obrazku dorysować basen:

```
230 CIRCLE (195,155),20,5, , ,1/3
240 PAINT (195,155),5,5
```

Nasz dom powinien oczywiście mieć komin, a także alejkę ogrodową:

```
130 LINE (105,130)-(155,192),1
140 LINE (115,130)-(165,192),1
150 LINE (155,192)-(165,192),1
160 LINE (105,130)-(115,130),1
170 LINE (70,25)-(76,45),6,BF
180 PAINT (160,190),1,1
```

Dokończymy nasz obrazek dodając okna, a zrobimy to posługując się instrukcją DRAW. Jest to najbardziej uniwersalna spośród wszystkich instrukcji graficznych i pozwala na tworzenie bardzo złożonych projektów. Jej działanie przypomina bardzo instrukcję PLAY, o której mówiliśmy w rozdziale 13. Tekst, który

występuje po słowie DRAW, zawiera wszystkie informacje niezbędne do wykonania instrukcji. Dopiszcie do programu linię:

```
190 DRAW    „C1;BM75,70;U8;D12;L6;R12;U12;L12;D12;U6;
            R12"
```

i w lewym górnym narożu pojawi się okno! Przeanalizujemy ten tekst krok po kroku, aby wyjaśnić jego działanie.

Parametr C określa kolor, w jakim ma być wykonany rysunek. Można go zmienić w dowolnym miejscu tekstu.

Parametr M, po którym następuje para współrzędnych (w naszym przykładzie 75,70), informuje, że należy wykonać ruch do wskazanego punktu. Znak „plus" lub „minus" umieszczony przed współrzędną oznacza, że ruch należy wykonać względem bieżącego położenia. Na przykład, gdyby w naszym tekście znajdował się fragment „BM+75,−70", jego sens byłby taki: należy wykonać ruch o 75 punktów obrazowych w prawo i o 70 w górę. Normalnie, po wykonaniu ruchu rysowana jest linia, która łączy nowo narysowany punkt z punktem poprzednim. Przed rozpoczęciem rysowania okna chcielibyśmy przesunąć się do nowego położenia bez rysowania czegokolwiek i dlatego poprzedziliśmy parametr M parametrem B. Wobec tego kolejna instrukcja będzie „pustym" ruchem, to znaczy, że zostanie on wykonany, ale na rysunku nic się nie zmieni. Zatem wyrażenie „BM75,70" oznacza przesunięcie do punktu o współrzędnych (75 70) bez pozostawiania śladu na ekranie.

Parametr U jest poleceniem wykonania ruchu w górę o wskazaną liczbę punktów obrazowych. Analogicznie D12 oznacza przesunięcie w dół ekranu o 12 punktów, L16 — w lewo o 16 punktów, a R12 — w prawo o 12 punktów obrazowych.

Podobnie jak w instrukcji PLAY średniki mają za zadanie poprawienie czytelności. Możemy używać zmiennych i do tekstu umieszczonego w instrukcji DRAW wstawiać gotowe teksty, zupełnie tak samo jak w instrukcji PLAY. Parametr X ułatwi nam narysowanie pozostałych trzech okien.

```
200 W$ = „U6; D12; L6; R12; U12; L12; D12; U6; R12"
210 DRAW „C1; BM145; 70; XW$"
220 DRAW „C1; BM75,100; XW$;"
230 DRAW „C1; BM145,100; XW$;"
```

Nasz obrazek jest już gotów. Instrukcja DRAW zawiera jednak szereg możliwości, o których jeszcze nie mówiliśmy. Poświęćmy więc jej nieco więcej czasu.

Używaliśmy parametrów U, L, D i R do wyznaczania konkretnych kierunków ruchu. Możemy też zadać ruch po prostej za pośrednictwem parametrów:

E ukośnie do góry i w prawo
F ukośnie w dół i w prawo
G ukośnie w dół i w prawo
H ukośnie do góry i w lewo

Przed każdym poleceniem w instrukcji DRAW możemy umieścić parametr N, który oznacza, że po wykonaniu instrukcji ma nastąpić powrót do położenia początkowego, na przykład:

```
10 SCREEN 3
20 DRAW „BM128, 96C1; NU80; C2NE80; C3N80"
30 DRAW „C5; NF80; C6; ND80; C7; NG80"
40 DRAW „C8; NL80; C9; NH80"
50 Z = INT (RND(1)*4)
60 DRAW „A = Z;"
70 GOTO 20
```

W linii 60 wprowadziliśmy jeszcze jeden parametr. Parametr A zmienia kąt ustawienia rysunku na ekranie. Z założenia wartość A wynosi zero. Jeżeli jest ona równa 1, rysunek zostaje przekręcony w prawo o ćwierć obrotu, jeżeli 2 — o pół obrotu, a jeżeli 3 — o trzy czwarte obrotu.

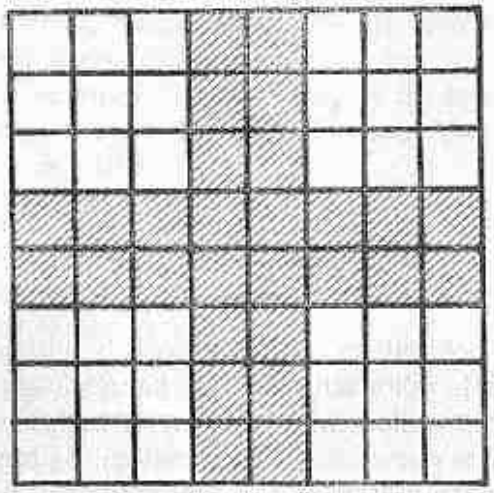
Możemy wreszcie zmieniać skalę rysunku przy udziale parametru S. Może on przybierać dowolną wartość w przedziale od 0 do 255 i określa skalę, w jakiej zostanie wykonany każdy fragment tekstu w instrukcji DRAW. Użycie parametru S4 jest równoważne pominięciu go.

16
Skrzaty

W rozdziale 14 pokazaliśmy, w jaki sposób możemy przesuwać obiekty po ekranie. Z możliwości tej korzysta się nie tylko w grach, ma ona duże znaczenie i w innych zastosowaniach. Basic MSX jest tu bardzo pomocny dzięki skrzatom. Są to po prostu bloki graficzne, których kształt można zdefiniować w programie. Raz określone, mogą zostać umieszczone na ekranie i być przesuwane po nim zupełnie dowolnie, zgodnie z życzeniem programisty. Gdybyśmy np. pisali program, w którym użytkownik kieruje statkiem kosmicznym, zdefiniowalibyśmy skrzata w kształcie tego właśnie statku. Zrobiwszy to, moglibyśmy latać nim po ekranie, używając w tym celu instrukcji Basica MSX, operujących na skrzatach, uruchamianych klawiszami sterującymi kursorem.

Podstawową sprawą jest zdefiniowanie skrzatów. Skrzaty mogą mieć 8 punktów obrazowych wysokości i 8 szerokości (8*8) albo 16 punktów obrazowych szerokości i 16 wysokości (16*16). Zacznijmy od zdefiniowania skrzata 8*8 w kształcie krzyża. Rys. 12 przedstawia obraz, który chcielibyśmy otrzymać. Na diagramie tym każdy kwadrat odpowiada jednemu punktowi obrazowemu naszego skrzata. Instrukcja, która umieszcza skrzata na ekranie, określa również jego kolor. Każdy punkt obrazowy będzie miał albo ów żądany kolor, albo barwę tła. Na rys. 12 zaciemnione kwadraty odpowiadają punktom obrazowym, które chcemy mieć zabarwione, a jasne kwadraty tym, które mają pozostać w kolorze tła.

Przekonamy się wkrótce, że w przypadku skrzatów wygodniej jest korzystać z dwójkowego lub szesnastkowego układu liczbowego niż z powszechnie stosowanego układu dziesiętnego. Jeżeli



Rys. 12. Skrzat 8×8 w kształcie krzyża

nie jesteście zaznajomieni z tymi układami liczbowymi, zajrzyjcie do *Dodatku B*, gdzie znajduje się ich omówienie.

Możemy też przedstawić skrzaty w postaci liczbowej. Oznaczmy jedynką punkt obrazowy w kolorze atramentu, a zerem — w kolorze tła.

```
00011000
00011000
00011000
11111111
11111111
00011000
00011000
00011000
```

Można chyba dostrzec na tym rysunku kształt krzyża? Zera i jedynki dokładnie odpowiadają rzeczywistej reprezentacji skrzata we wnętrzu komputera; z tego to właśnie powodu liczby dwójkowe są tak użyteczne — składają się wyłącznie z dwóch cyfr: 0 i 1. Komputer MSX traktuje każdą linię jako liczbę reprezentującą wzór tej linii. Poniżej widać, w jaki sposób dobiera się te liczby:

Liczby dwójkowe	Odpowiedniki dziesiętne
00011000	24
00011000	24
00011000	24
11111111	255
11111111	255
00011000	24
00011000	24
00011000	24

Po prostu sprawdzamy każdą linię projektu, by stwierdzić, jaka liczba dwójkowa jest przez nią reprezentowana. Następnie zmieniamy zapis tych liczb na dziesiętny i łączymy wszystkie wartości w jednej instrukcji SPRITE$:

```
SPRITE$ (1) = CHR$ (24)+CHR$ (24)+
CHR$ (24)+CHR$ (255)+
CHR$ (255)+CHR$ (24)+CHR$ (24)+CHR$ (24)
```

W ten sposób został zdefiniowany skrzat w kształcie krzyża. Liczba 1 ujęta w nawiasy oznacza, że jest to wzór skrzata numer 1.

Programowanie nie byłoby zabawą, gdybyśmy wciąż musieli zmieniać zapis liczby z dwójkowego na dziesiętny i odwrotnie. Dlatego też Basic MSX pomaga nam w tym.

Funkcja BIN$ daje w wyniku argument zapisany w postaci dwójkowej, na przykład:

```
PRINT BIN$ (34)
```

Podobnie HEX$ i OCT$ dają postać szesnastkową (heksadecymalną) i ósemkową (oktalną), na przykład:

```
PRINT HEX$ (12), OCT$ (48)
```

W Basicu MSX możemy korzystać również z liczb zapisanych w postaci dwójkowej, poprzedzając każdą z nich znakiem „&B", na przykład:

```
LET A = &B10011110
```

To samo odnosi się do liczb ósemkowych i szesnastkowych, które muszą być poprzedzone znakami &H i &O, na przykład:

```
PRINT &HOOFE, &0374
```

Dzięki temu życie staje się znacznie łatwiejsze. Poniżej wypisany jest program, który umieszcza naszego skrzata w kształcie krzyża pośrodku ekranu. Prześledźmy to krok po kroku.

```
 10 SCREEN 2,0
 20 FOR I = 1 TO 8
 30 READ A$
 40 A = VAL („&B"+A$)
 50 S$ = S$+CH$(A)
 60 NEXT I
 70 SPRITE$ (1) = S$
 80 PUT SPRITE 1, (128, 96), 1,1
 90 GOTO 90
100 DATA 00011000
110 DATA 00011000
120 DATA 00011000
130 DATA 11111111
140 DATA 11111111
150 DATA 00011000
160 DATA 00011000
170 DATA 00011000
```

W linii 10 nie tylko ustawiamy mod ekranu. Wykorzystujemy również drugi parametr instrukcji SCREEN w celu określenia, z którego rodzaju skrzatów będziemy korzystać. Parametr ten może przybierać jedną z czterech wartości:

0	8*8	nie powiększony
1	8*8	powiększony
2	16*16	nie powiększony
3	16*16	powiększony

Zdecydowaliśmy już, że w naszym przykładzie będziemy używali skrzatów 8*8, a zatem nadamy temu parametrowi wartość 0 lub 1. Różnica między skrzatem powiększonym i nie powiększonym wiąże się z wielkością obrazu na ekranie. W skrzatach powiększonych na każdy element przeznacza się więcej niż jeden punkt obrazowy. Różnice te dostrzegamy zmieniając linię 10 na:

```
SCREEN 2,1
```


Następnie definiujemy skrzata. Instrukcje DATA w liniach od 100 do 170 zawierają wszystkie potrzebne dane zapisane w postaci dwójkowej. Przyjrzyjcie się tej części programu; powinniście dojrzeć kształt krzyża, który nie byłby widoczny, gdyby dane zostały zapisane w układzie dziesiętnym. Liczby te muszą zostać wczytane i wprowadzone do instrukcji SPRITE$ (1). Jest to wykonane w liniach od 20 do 70. W linii 30 następuje przeczytanie jednej pozycji z instrukcji DATA. W linii 40 zostaje dopisane przed nią „&B", a otrzymany tekst zamienia się na liczbę. W linii 50 do zmiennej S$ dodaje się kolejną pozycję, która zawiera wszystkie potrzebne informacje. Ostateczna wartość S$ zostaje podstawiona pod SPRITE$ (1).

SPRITE$ (1) jest wzorem skrzata numer 1; możemy umieścić go na ekranie posługując się instrukcją PUT SPRITE. Instrukcja ta ma kilka parametrów. Pierwszy z nich jest numerem planu, ale zajmiemy się nim później. Drugi podaje współrzędne miejsca, w którym skrzat ma być umieszczony. Trzeci podaje kolor atramentu, a czwarty — numer wzoru. Zdefiniowaliśmy SPRITE$ (1), a zatem na czwartym miejscu wpiszemy 1.

Gdybyśmy chcieli zdefiniować skrzata o większych rozmiarach albo zawierającego więcej szczegółów, użylibyśmy skrzata o rozmiarach 16*16. W tym przypadku do zdefiniowania wzoru potrzebne są 32 pozycje danych. Pierwszych szesnaście pokazuje wzór lewej połówki skrzata, a kolejna szesnastka — wzór prawej połówki. Poniższy program przesuwa kwadrat po ekranie. Kwadrat jest skrzatem o wymiarach 16*16:

```
 10 C =1 : D = 1
 20 X = 128 : Y = 96
 30 SCREEN 3, 3
 40 FOR I = 1 TO 32
 50 READ A$
 60 A = VAL („&B"+A$)
 70 S$ = S$+CHR$ (A)
 80 NEXT I
 90 SPRITE$ (1) = S$
100 PUT SPRITE 1, (X, Y), 7
110 PUT SPRITE 3, (140, 100), 1, 1
```

```
120 PUT SPRITE 0, (30, 40), 6, 1
130 X = X+C
140 Y = Y+D
150 IF X = 220 OR X = 1 THEN C = −C: BEEP
160 IF Y = 160 OR Y = 0 THEN D = −D: BEEP
170 GOTO 100
180 DATA 11111111
190 DATA 11111111
200 DATA 11000000
210 DATA 11000000
220 DATA 11000000
230 DATA 11000000
240 DATA 11000000
250 DATA 11000000
260 DATA 11000000
270 DATA 11000000
280 DATA 11000000
290 DATA 11000000
300 DATA 11000000
310 DATA 11000000
320 DATA 11111111
330 DATA 11111111
340 DATA 11111111
350 DATA 11111111
360 DATA 00000011
370 DATA 00000011
380 DATA 00000011
390 DATA 00000011
400 DATA 00000011
410 DATA 00000011
420 DATA 00000011
430 DATA 00000011
440 DATA 00000011
450 DATA 00000011
460 DATA 00000011
470 DATA 00000011
480 DATA 11111111
490 DATA 11111111
```


Zwróćcie uwagę, co dzieje się, gdy poruszający się kwadrat mija dwa nieruchome kwadraty. Zdaje się on przesuwać p r z e d pierwszym i z a drugim. Dzieje się tak, ponieważ pierwszy parametr instrukcji PUT SPRITE umieścił te trzy skrzaty w trzech różnych planach. Plany mogą mieć numery od 0 do 31, przy czym zero oznacza plan najbardziej wysunięty „do przodu". Ustawiając skrzaty w różnych planach, możemy uzyskać w programach wspaniałe efekty.

Dopiszmy jeszcze kilka linii do naszego programu:

```
  3 ON SPRITE GOSUB 1000
  5 SPRITE ON
1000 BEEP
1010 PUT SPRITE 1, STEP (40, 40), 7,1
1020 RETURN
```

Następuje teraz sprawdzenie, czy doszło do zderzenia dwóch skrzatów. Jeżeli tak, linia 3 powoduje skok do procedury w linii 1000. Składnia tej instrukcji przypomina bardzo instrukcję ON...STOP; istnieją również instrukcje SPRITE OFF i SPRITE STOP, których działanie przypomina STOP OFF i STOP STOP. Zwróćcie uwagę na składnię instrukcji PUT SPRITE. Użycie STEP pozwala zdefiniować ruch względny. Skrzat zostaje przesunięty o 40 punktów obrazowych w prawo i o 40 poniżej dotychczasowego położenia, a nie do punktu o współrzędnych (40, 40).

Program zawiera wiele instrukcji DATA, w których przechowuje wszystkie dane w postaci dwójkowej. W bardziej złożonych programach, ze względu na oszczędność miejsca, wygodniej jest przejść do zapisu szesnastkowego. Jest to łatwiejsze niż przejście do zapisu dziesiętnego, na przykład:

```
10 SCREEN 2, 2
20 A$ = „ " : FOR I = 1 TO 32
30 READ SP$ : A$ = A$+CHR$(VAL(„&H"+SP$))
40 NEXT I : SPRITE$ (0) = A$
50 SPRITE ON
60 FOR Y2 = 170 TO 10 STEP −20
70 FOR Y1 = 150 TO 10 STEP −20
80 FOR X2 = 240 TO 10 STEP −10
```

```
 90 FOR X1 = 220 TO 10 STEP -10
100 PUT SPRITE 0, (X1, Y1), 8, 0
110 PUT SPRITE 1, (X2, Y2), 11, 0
120 NEXT X1, X2, Y1, Y2
130 GOTO 60
140 DATA 03, 0F, 1F, 39, 79, FF, FF, FF
150 DATA 2A, 2A, 2A, 4A, 4A, 52, 92, 92
160 DATA C0, F0, F8, 9C, 9F, FF, FF, FF
170 DATA 54, 54, 54, 52, 52, 4A, 49, 49
180 END
```

Istnieje inny jeszcze, bardzo przydatny zbiór instrukcji, które pomagają manipulować skrzatami. Są to instrukcje ON...STRIG GOSUB. Za pomocą instrukcji STRIG(n) ON, STRIG(n) OFF, STRIG(n) STOP I ON...STRIG GOSUB sprawdza się, czy został naciśnięty przycisk „joysticka" (manetki sterowniczej). Jeżeli tak, następuje skok do podprogramu. Argument (n) może przybierać wartości od 0 do 4. Jeżeli n = 0, w charakterze przycisku używa się klawisza spacji, na przykład:

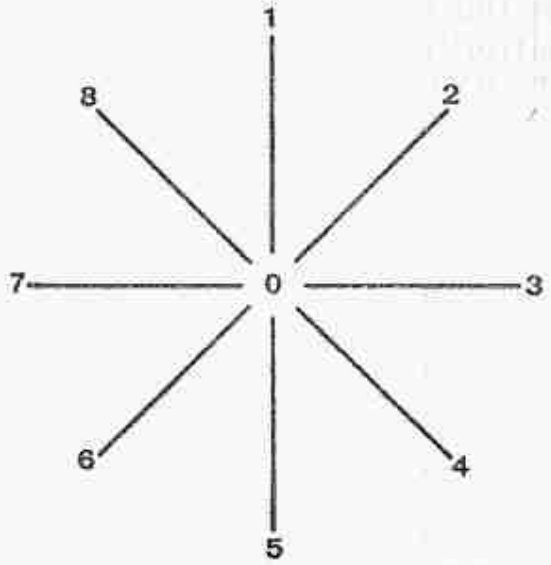
```
10 STRIG(0) ON
20 ON STRIG GOSUB 100
30 GOTO 20
100 BEEP
110 PRINT STRIG(0)
120 RETURN
```

Funkcja STRIG w linii 110 podaje aktualny stan przycisku manetki sterowniczej (czy został on naciśnięty, czy też nie).

Funkcja STICK również wiąże się z użyciem „joysticka". Podaje ona kierunek nachylenia drążka. Argument jej musi być równy 0, 1 lub 2. Jeżeli jest równy zeru, w charakterze manetki wykorzystuje się klawisze sterujące kursorem. Rys. 13 podaje wartości, jakie przybiera funkcja STICK.

Poniżej podajemy program, w którym wykorzystuje się funkcję STICK do kierowania ruchem skrzata.

```
10 REM KOMPOZYCJA EKRANU
20 X = 128 : Y = 96
30 SCREEN 2,2
```

Rys. 13. Wartości, jakie przybiera funkcja STICK

```
 40 A$ = " " : FOR B = 1 TO 32
 50 REM CZYTANIE WZORCA SKRZATA
 60 READ B$ : A$ = A$+CHR$(VAL("&B"+B$))
 70 NEXT B : SPRITE$(0) = A$
 80 REM WYŚWIETLANIE SKRZATA
 90 PUT SPRITE 0, (X,Y), 11
100 REM WYKORZYSTANIE FUNKCJI STICK DO KIEROWANIA SKRZATEM
110 S = STICK(0) : IF S = 0 THEN 110
120 REM SKOK DO PROCEDUR GENERUJĄCYCH RUCH
130 ON (S+1)/2 GOTO 500,530,560,590
140 REM WZORZEC SKRZATA
150 REM DOLNA CZĘŚĆ SKRZATA
160 DATA 11110000
170 DATA 11110000
180 DATA 00001111
190 DATA 00000111
200 DATA 00000011
210 DATA 00000111
220 DATA 00001111
```

```
230 DATA 11110000
240 DATA 11110000
250 DATA 00001111
260 DATA 00000111
270 DATA 00000011
280 DATA 00000111
290 DATA 00001111
300 DATA 11110000
310 DATA 11110000
320 REM GÓRNA CZĘŚĆ SKRZATA
330 DATA 00000000
340 DATA 00000000
350 DATA 00000000
360 DATA 00000000
370 DATA 00000000
380 DATA 00000000
390 DATA 11100000
400 DATA 11110000
410 DATA 11110000
420 DATA 11100000
430 DATA 00000000
440 DATA 00000000
450 DATA 00000000
460 DATA 00000000
470 DATA 00000000
480 DATA 00000000
490 REM DO GÓRY
500 Y = Y-1 : IF Y = -1 THEN Y = 96
510 GOTO 90
520 REM W PRAWO
530 X =X+1 : IF X = 256 THEN X = 128
540 GOTO 90
550 REM DO DOŁU
560 Y = Y+1 : IF Y = 192 THEN Y = 96
570 GOTO 90
580 REM W LEWO
590 X = X-1 : IF X = -1 THEN X = 128
600 GOTO 90
```


17

Rozmaitości

Tak więc omówiliśmy wszystkie podstawowe zagadnienia dotyczące Basica MSX. Posiadacie już niezbędne umiejętności, które pozwalają na napisanie programów korzystających z grafiki, dźwięku oraz innych udogodnień, jakie daje komputer MSX. Basic MSX dysponuje jednakże wielkim bogactwem funkcji i instrukcji, o których jeszcze nie mówiliśmy. Zrobimy to teraz.

Po pierwsze, przyjrzyjmy się, w jaki sposób wprowadza się teksty z klawiatury. Wiemy, że w jednej instrukcji INPUT możemy umieścić więcej niż jedną zmienną:

```
10 input „trzy nazwiska"; a$,b$,c$
20 print a$,b$,c$
```

Dowiedzieliśmy się, że w takim przypadku wprowadzamy nazwiska oddzielnie, naciskając na końcu każdego z nich klawisz RETURN. Możemy jednak postąpić inaczej, pisząc od razu wszystkie trzy nazwiska i oddzielając je przecinkami. Klawisz RETURN należy nacisnąć dopiero po napisaniu trzeciego z nich, a wówczas komputer podstawi każde nazwisko pod odpowiednią zmienną. Jest to bardzo wygodne rozwiązanie, pod tym wszakże warunkiem, że w żadnym z nazwisk nie chcemy umieścić przecinka. Jako pierwsze nazwisko w kolejnym programie spróbujcie wprowadzić „Kowalski, Jan". Pojawi się komunikat „Extra ignored" (dalszy ciąg pominięty), a instrukcja w linii 20 da dość dziwny wynik:

```
10 input „nazwisko"; a$
20 print a$
```

Komputer założył po prostu, że przecinek w pierwszym nazwisku oznacza koniec pierwszej danej. Ponieważ instrukcja

INPUT w linii 10 nie przewiduje kolejnych danych, pozostała część nazwiska została pominięta. Może to powodować nieprzewidziane kłopoty, dlatego wprowadzimy jeszcze jedną instrukcję wejścia — LINE INPUT. Instrukcja ta sprawia, że czytany jest cały tekst, aż do naciśnięcia klawisza RETURN. Cały przeczytany tekst zostanie przypisany wskazanej zmiennej, pod warunkiem, że nie zawiera on więcej niż 255 znaków.

```
10 cls
20 print „wprowadź długi tekst"
30 print „zawierający przecinek"
40 line input a$
50 print a$
```

Uniwersalnym rozszerzeniem instrukcji PRINT jest PRINT USING. Wprowadza ono dużą różnorodność sposobów prezentowania tekstów i liczb na ekranie. Istnieje wiele wersji instrukcji PRINT USING, co pozwala na różne formatowanie tekstów i liczb, na przykład:

```
print using „!";
```

powoduje wydrukowanie tylko pierwszego znaku danego tekstu:

```
10 x$ = „ciasto"
20 print using „!"; x$
```

Istnieją pewne odmiany tej instrukcji stosowane do formatowania liczb. Z pomocą znaku # określamy sposób, w jaki liczba ma być przedstawiona na ekranie. Każdy znak # reprezentuje jedną cyfrę.

```
print using „###.##"; 12.4, 86.732, .248, 16.86, 3.00
```

Kropka „." we wzorcu pokazuje, w którym miejscu wydruku ma się znajdować kropka dziesiętna. Jeżeli wzorzec przewiduje, że przed kropką dziesiętną pojawi się cyfra, będzie ona zawsze drukowana, nawet gdy będzie to zero.

Jeżeli chcemy, aby drukowany był również znak (+ lub −) liczby, musimy na początku wzorca umieścić znak +:

```
print using „+###.##";−4879.3
```


Możemy również zażądać, aby liczby ujemne były drukowane ze znakiem −, dopisując ten znak na końcu wzorca:

```
print using „###.##−";−4879.3
```

Jeżeli na początku wzorca umieścimy gwiazdki „**", poprzedzające liczbę spacje zostaną zastąpione gwiazdkami:

```
print using „**###.##";3.46,−3.46
```

Umieszczenie podwójnego znaku funta (£) sprawi, że będzie on drukowany na początku tak sformatowanej liczby:

```
print using „££###.##";83.64;83.64,−83.64
```

Użycie kombinacji „**£" łączy powyższe dwa efekty. Na początku liczby zostanie wydrukowany znak funta, a poprzedzające ją spacje zostaną zastąpione gwiazdkami.

Śledzenie błędów

Jeżeli komputer napotka w programie linię, której z jakiegokolwiek powodu nie będzie mógł wykonać, wyświetli na ekranie komunikat o błędzie. Komunikaty te są tak pomyślane, aby dać nam pewne pojęcie o naturze problemu. Lista możliwych komunikatów wraz z kodami błędów znajduje się w *Dodatku A*. Kod błędu może sięgać 255; widzicie zatem, że Basic nie wykorzystuje dużej liczby tych kodów. Basic MSX umożliwia pisanie własnych procedur śledzenia błędów. Widać to najlepiej na poniższym przykładzie:

```
  10 ON ERROR GOTO 1000
  20 INPUT WIEK
  30 IF WIEK < 0 OR WIEK > 130 THEN ERROR 255
  40 GOTO 10
1000 IF ERR = 255 THEN PRINT „Poza dozwolonym zakresem"
1010 RESUME 10
```

Ten prosty program pyta o wiek użytkownika i wyświetla go

na ekranie. Korzysta przy tym z procedury śledzenia błędów w celu zabezpieczenia przed bezsensownymi odpowiedziami. Założenie, że wiek użytkownika nie będzie mniejszy od zera ani większy od 130, jest rozsądne. Program stwierdza, że jeżeli zostanie napotkana liczba spoza tego zakresu, będzie to potraktowane jako błąd.

Linia 10 nakazuje, by w razie wystąpienia jakiegokolwiek błędu został wykonany skok do procedury śledzenia błędów, która rozpoczyna się w linii 1000. Linia 30 sprawdza poprawność wczytanego wieku. W razie napotkania wartości nie do przyjęcia, w linii tej zostanie stwierdzone, że pojawił się błąd o kodzie 255. Powiedzieliśmy już, że w razie napotkania błędu musi nastąpić skok do linii 1000 (linia 10), i tak właśnie się dzieje.

Warunek w linii 1000 jest spełniony i dlatego na ekranie pojawi się komunikat „poza dozwolonym zakresem". Każda procedura śledzenia błędów musi się kończyć instrukcją RESUME (wznawiać), która informuje komputer, od którego miejsca ma wznowić wykonanie programu. W tym przypadku ma to być linia 10.

Poniżej mamy wszystkie słowa kluczowe związane ze śledzeniem błędów.

ON ERROR	używane jest razem z GOTO i wskazuje początek procedury śledzenia błędów.
ERROR n	informuje, że wystąpił błąd o kodzie n.
ERR	zmienna Basica MSX, która przechowuje bieżący kod błędów.
ERL	również zmienna Basica MSX, zawierająca numer linii, w której został wykryty błąd.
RESUME	instrukcja wskazująca miejsce, w którym ma nastąpić wznowienie wykonywania programu. Występuje ona w czterech formatach:
RESUME lub RESUME 0	wykonanie programu zostaje wznowione od linii, w której wystąpił błąd.

RESUME NEXT	wykonanie programu zostaje wznowione od instrukcji znajdującej się bezpośrednio za instrukcją, która spowodowała błąd.
RESUME n	wykonanie programu zostaje wznowione od linii o wskazanym numerze.

Jeżeli instrukcja RESUME zostanie napotkana poza procedurą śledzenia błędu, spowoduje to błąd. Zachowajcie ostrożność w trakcie konstruowania programu, aby nie dopuścić do przypadkowego „wpadnięcia" na instrukcję RESUME.

Dobierając kod błędu dobrze jest wykorzystywać wysokie numery, powiedzmy z przedziału 230—255. Chodzi o to, że niższe numery mogą być wykorzystane do komunikatów o błędach w trakcie późniejszego rozszerzania standardu Basica MSX. Użycie wyższych numerów gwarantuje, że wasze programy pozostaną zgodne ze standardem Basica MSX.

W rozdziale 1 omówiliśmy pokrótce kod maszynowy. Przypomnijmy, że w rzeczywistości komputer rozumie tylko te programy, które składają się z instrukcji używanych przez mikroprocesor. Są to tak zwane programy w kodzie maszynowym. Wszystkie nasze programy napisane w Basicu, zanim komputer przystąpi do ich wykonywania, muszą zostać przetłumaczone przez interpreter Basica na kod maszynowy. Programy napisane w kodzie maszynowym są to długie ciągi liczb umieszczone w pamięci komputera. Inna część pamięci przeznaczona jest do przechowywania liczb odpowiadających znakom wyświetlanym na ekranie telewizora. Byłoby dobrze móc od czasu do czasu posłużyć się zawartością tych obszarów. Omawianie programowania w kodzie maszynowym wykracza poza zakres tej książki, ale powiemy o instrukcjach, które pozwalają pracować na poziomie kodu maszynowego.

Prawdopodobnie najważniejszą z nich jest POKE. Umożliwia nam ona wpisanie konkretnej liczby pod wskazany adres pamięci RAM:

```
poke 35386,38
```

Liczba 38 zostanie wpisana do pamięci pod adres 35386. Sprawdźcie w podręczniku, jak wygląda podział pamięci RAM w

waszym komputerze. Korzystajcie z instrukcji POKE z dużą ostrożnością, jeżeli bowiem nie jesteście pewni, co robicie, możecie łatwo „rozbić" program. Program może się niespodziewanie zawiesić albo ekran wypełni się nagle dziwnymi znakami. Nie uszkodzi to wprawdzie komputera, ale skutecznie popsuje wam humor, jeżeli przez poprzednie pół godziny pracowicie wpisywaliście treść programu, a teraz okazało się, że nie można go odzyskać!

Funkcja PEEK pozwala badać zawartość dowolnej komórki pamięci:

```
print peek(384)
```

Powyższy przykład spowoduje wydrukowanie zawartości komórki pamięci o adresie 384.

W pamięci wydzielony jest specjalny obszar (tzw. pamięć ekranu), przeznaczony do przechowywania zawartości ekranu podczas pracy w trybie graficznym. Basic MSX używa dwóch podobnych słów kluczowych, które operują na pamięci ekranu. Lewy górny narożnik ekranu odpowiada adresowi 1 w pamięci ekranu, punkt z prawej strony ma adres 2, następny 3 i tak dalej, aż do osiągnięcia dołu ekranu. Poniższy program używa instrukcji VPOKE do wyświetlania na ekranie przypadkowych znaków:

```
10 screen 3
20 for a = 1 to 12800
30 b = int(rnd(1)*255)
40 vpoke a,b
50 next a
60 goto 10
```

VPEEK umożliwia badanie zawartości pamięci ekranu:

```
print vpeek(373)
```

Jeszcze trochę o funkcjach

Basic MSX dysponuje ogromnym bogactwem funkcji. Z wieloma z nich już się zetknęliście, ale pozostały nam jeszcze niektóre do omówienia:

abs(x)	przybiera wartość bezwzględną x, czyli wartość argumentu z pominięciem znaku. Na przykład: print abs(8); abs(−14.7)
ant(x)	jedna spośród szeregu funkcji trygonometrycznych, interesująca tylko dla tych, którzy zajmują się matematyką. Przybiera ona wartość arcusa tangensa argumentu, wyrażoną w radianach. Zauważcie, że wartości wszystkich odwrotnych funkcji trygonometrycznych są w Basicu MSX wyrażone w radianach, a nie w stopniach.
cos(x)	przybiera wartość cosinusa argumentu wyrażonego w radianach.
csrlin	wskazuje pionową współrzędną kursora na ekranie.
exp(x)	przybiera wartość stałej e (uniwersalna stała matematyczna) podniesionej do potęgi równej argumentowi.
fre(0)	podaje wielkość obszaru pamięci dostępnej dla Basica. Argumentem może być dowolna liczba, na przykład: print fre(0); fre(10); fre(200)
fre(„")	podaje wielkość obszaru pamięci przeznaczonego dla tekstów. Argumentem może być dowolny tekst, na przykład: print fre(„"); fre („msx")
pos(a)	position (położenie) = podaje bieżące położenie kursora.
sgn(x)	przybiera wartość 1, jeżeli argument jest dodatni, lub −1 jeżeli argument jest ujemny.
sin(x)	przybiera wartość równą sinusowi kąta wyrażonego w radianach.
tan(x)	przybiera wartość równą tangesowi kąta wyrażonego w radianach.

time — czas, godzina — komputer MSX wyposażony jest w wewnętrzny zegar, którego stan co jedną pięćdziesiątą sekundy automatycznie zwiększa się o jeden. Funkcja TIME podaje bieżący stan zegara. Możecie używać jej wraz z funkcją RND do generowania liczb w pełni losowych.

I to już wszystko!

Od tej chwili jesteście zdani na siebie. Powinniście w pełni rozumieć Basic MSX i dlatego proponujemy wam eksperymentowanie dla własnej przyjemności. A nam pozostaje już tylko podać kilka informacji uzupełniających, które znajdują się w dodatkach, oraz życzyć wam powodzenia w programowaniu komputera MSX.

Dodatek A
Komunikaty o błędach

Kod	Komunikat	Znaczenie
1	NEXT without FOR	Zmienna w instrukcji NEXT bez otwierającej instrukcji FOR.
2	Syntax error	Niewłaściwy ciąg znaków albo instrukcja z błędem ortograficznym.
3	RETURN without GOSUB	Instrukcja RETURN nie poprzedzona instrukcją GOSUB.
4	Out of DATA	Za mała ilość danych w instrukcji DATA w trakcie wykonywania instrukcji READ.
5	Illegal function call	Argument funkcji matematycznej albo tekstowej wykracza poza dozwolony zakres.
6	Overflow	Wynik obliczeń ma wartość zbyt dużą, by można ją było zinterpretować.
7	Out of memory	Program albo dane zbyt obszerne.
8	Undefined line number	Odwołanie do nie istniejącej linii.
9	Subscript out of range	Odwołanie do elementu tablicy, którego indeks wykracza poza przedział zdefiniowany instrukcją DIM.
10	Redimensioned array	Dwukrotne zdefiniowanie tej samej tablicy.
11	Division by zero	Nastąpiło dzielenie przez zero.
12	Illegal direct	Niedozwolona instrukcja w trybie konwersacyjnym.
13	Type mismatch	Tekst przypisany zmiennej liczbowej lub odwrotnie.

14	Out of string space	Basic nie dysponuje już wolnym miejscem w obszarze pamięci przeznaczonym dla tekstów.
15	String too long	Długość tekstu przekracza 255 znaków.
16	Formula too complex	Wyrażenie tekstowe jest albo zbyt długie, albo zbyt skomplikowane.
17	Can't continue	Kontynuacja programu nie jest możliwa ze względu na błąd albo na modyfikację.
18	Undefined user	Wywołanie funkcji nie poprzedzone odpowiednią instrukcją DEF FN
19	Device I/O error	Wystąpił błąd we/wy w trakcie obsługi magnetofonu, drukarki albo monitora.
20	Verify error	Program znajdujący się w pamięci różni się od zapisanego na kasecie.
21	No RESUME	Procedura śledzenia błędów nie zawiera RESUME.
22	RESUME without error	Instrukcja RESUME została napotkana przed wejściem do procedury śledzenia błędów.
23	Unprintable error	Niezdefiniowany komunikat o błędzie.
24	Missing operand	Wyrażenie, w którym brakuje operandu.
25	Line buffer overflow	Zbyt wiele znaków w linii.
26—49		Zarezerwowane dla późniejszej rozbudowy Basica.
50	Field overflow	Instrukcja FIELD próbuje ulokować więcej bajtów niż zdefiniowano.
51	Internal error	Błąd w działaniu komputera.
52	Bad file number	Odwołanie do numeru zbioru, który nie został otwarty instrukcją OPEN albo znajduje się poza zakresem.
53	File not found	Instrukcja odwołuje się do zbioru, który nie znajduje się w pamięci.

54	File already open	Instrukcja OPEN operuje na zbiorze uprzednio otwartym.
55	Input past end	Użycie instrukcji INPUT po wprowadzeniu wszystkich danych do zbioru.
56	Bad file name	Niedozwolona nazwa w instrukcji OPEN itd.
57	Direct statement	Instrukcja w trybie konwersacyjnym napotkana w trakcie czytania z taśmy.
58	Sequential I/O only	Instrukcja operująca wyłącznie na zbiorze o dostępie swobodnym została użyta w odniesieniu do zbioru o dostępie sekwencyjnym.
59	File not OPEN	Wskazany zbiór nie został otwarty.
60—255		Zarezerwowane dla późniejszej rozbudowy Basica.

Dodatek B
Rachujemy wraz z komputerem

W naszym codziennym życiu przyzwyczailiśmy się do liczenia w układzie dziesiętnym. Używamy do tego dziesięciu symboli:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Jeżeli chcemy liczyć powyżej dziesięciu, piszemy więcej niż jedną kolumnę symboli, na przykład:

28

Znaczy to, że mamy 8 jednostek plus dwa pełne zbiory dziesięciu jednostek (2*10)+8. Jeżeli dwie kolumny nie wystarczają, możemy dodawać dalsze kolumny, z których każda reprezentuje wartości dziesięć razy większe niż kolumna na prawo od niej:

7834 = 7 tysięcy+8 setek+3 dziesiątki+4 jedności

Nie istnieje żaden sensowny powód, dla którego mielibyśmy liczyć dziesiątkami. Możemy również dobrze liczyć ósemkami używając ośmiu symboli:

0, 1, 2, 3, 4, 5, 6, 7

Gdybyśmy mieli więcej niż osiem elementów, moglibyśmy dopisywać dalsze kolumny tak samo, jak robiliśmy to w przypadku układu dziesiętnego, z tą wszakże różnicą, że kolejne kolumny reprezentowałyby teraz wartości osiem razy większe niż kolumny znajdujące się na prawo od nich. Jest to tak zwany u k ł a d ó s e m k o w y (o k t a l n y lub o c t). Na przykład:

674 = (6*8*8)+(7*8)+4
(ósemkowo) (dziesiętnie)

Możemy też liczyć dwójkami, a wtedy potrzebne są nam tylko dwa symbole:

0, 1

Jest to u k ł a d d w ó j k o w y (b i n a r n y lub b i n). W układzie dwójkowym każda kolumna reprezentuje wartości dwa razy większe niż kolumna znajdująca się na prawo od niej, na przykład:

1101 = (1*8)+(1*4)+(0*2)+1
(dwójkowo) (dziesiętnie)

Możemy nawet liczyć dwunastkami, piętnastkami, jak tylko chcemy, ale wtedy potrzebne są nam dodatkowe symbole, ponieważ dotychczasowe — od 0 do 9 — już nie wystarczają. Moglibyśmy użyć dowolnych znaków, ale najwygodniejsze chyba są litery alfabetu. Przypuśćmy, że chcielibyśmy liczyć szesnastkami. Jest to u k ł a d s z e s n a s t k o w y (h e x a d e c y m a l n y, albo po prostu h e x). Potrzeba nam sześciu dodatkowych symboli, a zatem używamy liter A, B, C, D, E, F:

Hex	Odpowiednik dziesiętny
A	10
B	11
C	12
D	13
E	14
F	15

Jeżeli mamy więcej niż F elementów (piętnaście w układzie dziesiętnym), znów są nam potrzebne dodatkowe kolumny:

8E = (8*16)+14
(hex) (dziesiętnie)

B43A = (11*16*16*16)+ (4*16*16)+(3*16)+10
(hex) (dziesiętnie)

Pokazaliśmy układy: dwójkowy, ósemkowy i szesnastkowy, ponieważ są one z powodzeniem wykorzystywane w komputerach. Jedna jednostka pamięci komputera (bajt) składa się z ośmiu bitów.

Każdy z tych bitów może zawierać wartości 0 lub 1 — inna możliwość nie istnieje. A to oznacza, że układ dwójkowy idealnie nadaje się do współpracy z komputerem. Wyobraźcie sobie jeden bajt, który znajduje się w takim oto stanie:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
1	0	0	1	1	1	0	0

Możemy zastąpić go liczbą dwójkową 10011100.

Jedyny problem polega na tym, że liczby dwójkowe są raczej niewygodne w użyciu, ponieważ stanowią długie ciągi jedynek i zer. Czasami lepiej jest przekształcić je do postaci dziesiętnej, którą łatwiej jest operować. To jednak rodzi nowe problemy, ponieważ trudno jest powiązać liczby dziesiętne z ich dwójkowymi odpowiednikami. Znacznie lepiej jest używać liczb ósemkowych albo szesnastkowych, ponieważ zarówno 8, jak i 16 są potęgami 2. Dzięki temu łatwiej jest dokonać przeliczenia z jednego układu na drugi. Układ szesnastkowy jest dziś najczęściej stosowany.

Dodatek C
Tabela kodów w ASCII

Kod	Klawisz	Kod	Klawisz	Kod	Klawisz
32	spacja	64	@	96	
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	—	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	Ø	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y

Kod	Klawisz	Kod	Klawisz	Kod	Klawisz
58	:	90	Z	122	z
59	;	91	[	123	£
60	<	92	\	124	|
61	=	93	]	125	§
62	>	94	^	126	~
63	?	95	—		

Słowniczek

(zestawił Andrzej Reich)

ABS — funkcja przybierająca wartość argumentu z pominięciem znaku

ASC — funkcja przypisująca znakowi jego kod ASCII

ATN — funkcja trygonometryczna, przybierająca wartość arcusa tangensa argumentu, wyrażona w radianach

ASCII — American Standard Code for Information Interchange — Amerykański Standardowy Kod dla Wymiany Informacji

AUTO — rozkaz powodujący automatyczne generowanie numeru linii

BACK SPACE — cofa kursor o jedno miejsce, kasując mijany znak

BASIC — Basic All-purpose Symbolic Instruction Set — Symboliczny Uniwersalny Kod dla Szkolenia Początkujących

BEEP — instrukcja włączająca brzęczyk

BIN$ — funkcja dająca argument zapisany w systemie binarnym (dwójkowym)

BLOAD — rozkaz ładowania z kasety binarnej wersji programu

BSAVE — rozkaz zapisania na kasecie treści programu

CAPS — blokada zmieniacza w klawiaturze komputera

CAS — CASsette — informacja, że zbiór danych zawarty jest w kasecie

CHR$ — funkcja zmieniająca argument (kod ASCII) na odpowiadający mu znak

CIRCLE — instrukcja rysowania okręgów

CLEAR — instrukcja, która pozwala określić, ile bajtów przeznacza się na teksty

CLOAD — rozkaz ładowania z kasety treści programu

CLOSE — instrukcja, która informuje komputer, że ma zamknąć zbiór

CLS — **CL**ear **S**creen — instrukcja, która usuwa z ekranu wszystkie napisy i ustawia kursor w pozycji początkowej

COLOR — colour — instrukcja definiująca barwy na ekranie komputera

CONT — **cont**inue — rozkaz powodujący wznowienie wykonywania zatrzymanego programu

COS — funkcja trygonometryczna, która przybiera wartość cosinusa argumentu wyrażonego w radianach

CRT — **C**athode **R**ay **T**ube — ekran telewizora

CSAVE — **C**assette **SAVE** — instrukcja, która powoduje zarejestrowanie treści programu na taśmie kasety magnetofonu

CSRLIN — funkcja wskazująca pionową współrzędną kursora na ekranie

CTRL — **C**on**TR**o**L** key — klawisz sterujący

CTRL Z — znak końca zbioru

DATA — instrukcja zawierająca listę danych, które mają być przeczytane przy użyciu instrukcji READ

DEF FN — instrukcja definiowania funkcji

DELETE — rozkaz pozwalający usunąć z programu dowolną linię lub grupę linii

DIM — dimension — instrukcja, która rezerwuje w pamięci miejsce dla tablicy

DRAW — instrukcja graficzna pozwalająca na tworzenie bardzo złożonych kompozycji

ELSE — rozszerzenie instrukcji warunkowej IF...THEN

END — instrukcja powodująca zatrzymanie programu

EOF — **E**nd **O**f **F**ile — funkcja, która przybiera wartość 1, jeśli został napotkany koniec zbioru

ERASE — instrukcja wymazująca zawartość tablic

ERL — zmienna Basica MSX zawierająca numer linii, w której został wykryty błąd

ERR — **ER**Ror — zmienna Basica MSX, która przechowuje bieżący kod błędu

EXP — funkcja, która przybiera wartość stałej e podniesionej do potęgi równej argumentowi

FOR...TO...STEP — instrukcja pętli

FRE — obszar przeznaczony dla Basica

GOSUB — instrukcja skoku do procedury rozpoczynającej się w linii o podanym numerze

GOTO — instrukcja skoku do linii o podanym numerze

GRP — **GR**a**P**hic screen — ekran graficzny

HEX$ — funkcja, która zamienia argument w postać szesnastkową

HOME/CLS — klawisz, który przesuwa kursor do położenia początkowego oraz czyści ekran

IF...THEN — instrukcja warunkowa

INKEY$ — instrukcja sprawdzająca, czy nie został naciśnięty jakiś klawisz

INPUT — instrukcja czytania danych

INPUT$ — instrukcja czytania zadanej liczby znaków bez wyświetlania ich na ekranie

INPUT/=/ — instrukcja czytania ze zbioru o podanym numerze

INSTR — funkcja, która pozwala szukać danego tekstu we wnętrzu innego tekstu

INT — funkcja, która podaje część całkowitą argumentu

KEY ON/OFF — instrukcja powodująca wyświetlenie/zgaszenie definicji klawiszy funkcyjnych

LEFT$ — funkcja, która wybiera zadaną liczbę znaków z lewej strony tekstu

LET — instrukcja podstawienia

LINE — instrukcja rysująca linię prostą między dwoma punktami

LINE INPUT — instrukcja czytania zawartości całej linii

LIST — rozkaz wyświetlenia na ekranie treści programu

LOCATE — instrukcja umieszczająca kursor w dowolnym miejscu ekranu

LPT — **L**ine **P**rin**T**er — drukarka

MAXFILES — instrukcja określająca maksymalną liczbę zbiorów, które mogą być równocześnie otwarte

MERGE — instrukcja pozwalająca dopisać do pamięci linie programu zapisanego na taśmie magnetofonu w kodzie ASCII

MID$ — funkcja wydobywająca ciąg znaków z dowolnego miejsca wewnątrz tekstu

MOTOR ON/OFF — instrukcja włączająca/wyłączająca silnik magnetofonu

NEW — instrukcja wymazująca program z pamięci komputera

NOT — operator zmieniający wartość logiczną na przeciwną
OCT$ — funkcja dająca w wyniku argument napisany w postaci oktalnej (ósemkowej)
ON ERROR — instrukcja śledzenia błędów
ON GOSUB — instrukcja skoku do jednej z procedur, wskazanej przez zmienną sterującą
ON GOTO — instrukcja skoku do jednej z linii wskazanej przez zmienną sterującą
ON KEY GOSUB — instrukcja skoku do jednej z procedur wskazanej przez klawisz funkcyjny
ON SPRITE — instrukcja śledząca kolizję dwóch skrzatów
OPEN — instrukcja otwierająca zbiór
OR — operator logiczny
OUTPUT — polecenie przesłania informacji do zbioru wyjściowego
PAINT — instrukcja, która powoduje zamalowanie dowolnego kształtu
PEEK — instrukcja czytająca zawartość dowolnej komórki pamięci
PLAY — instrukcja programująca muzykę w Basicu MSX
POINT — funkcja, która podaje kolor punktu o podanych współrzędnych
POKE — instrukcja umożliwiająca wpisanie konkretnej liczby pod wskazany adres pamięci RAM
POS — **POS**ition — funkcja, która podaje aktualne położenie kursora
PRESET — instrukcja, która punktowi obrazowemu o podanych współrzędnych przywraca barwy tła
PRINT — drukuj
PRINT # — instrukcja przesyłająca informacje do magnetofonu
PRINT USING — instrukcja wprowadzająca informację o podanym formacie
PSET — instrukcja nadająca punktowi obrazowemu żądaną barwę
PUT SPRITE — instrukcja umieszczająca skrzata we wskazanym miejscu na ekranie
RAM — **R**andom **A**kcess **M**emory — pamięć o dostępie swobodnym, przechowująca informacje tylko do momentu wyłączenia komputera
READ — instrukcja czytania danych zapisanych w instrukcji DATA
REM — **REM**ark — instrukcja, która wprowadza do treści programu komentarz opisujący wszystkie istotne szczegóły
RENUM — **RENUM**ber — rozkaz polecający zmienić numery linii całego programu lub jego części
RESTORE — instrukcja czytania instrukcją READ pierwszej pozycji na liście instrukcji DATA
RESUME — instrukcja wskazująca miejsce, w którym ma nastąpić wznowienie programu
RETURN — instrukcja powrotu do procedury lub klawisz powrotu karetki
RIGHT$ — funkcja wyodrębniająca znaki z prawej strony tekstu
RND — **R**a**ND**om — funkcja, która generuje w sposób losowy
ROM — **R**ead **O**nly **M**emory — pamięć stała, która zachowuje zawarte w niej informacje po wyłączeniu komputera
RUN — rozkaz wykonywania przez komputer programu począwszy od linii o najniższym numerze, kolejno, instrukcja po instrukcji
SAVE — rozkaz zapisania na taśmie programu w postaci tekstowej
SCREEN — instrukcja ustawiania ekranu w trybie graficznym lub tekstowym
SGN — funkcja, która przybiera wartość 1, jeżeli argument jest dodatni lub −1, jeżeli jest ujemny
SHIFT — klawisz zmieniacza
SIN — funkcja, która przybiera wartość równą sinusowi kąta wyrażonego w radianach
SOUND — instrukcja ustawiania zawartości rejestru generatora dźwięków
SPC — **SP**a**C**e — spacja, odstęp
SPRITE ON/OFF — włączenie/wyłączenie śledzenia kolizji skrzatów
SPRITE STOP — instrukcja, która powoduje zapamiętanie faktu kolizji dwóch skrzatów do chwili wywołania SPRITE ON
SPRITE$ — instrukcja definiująca skrzata

SQR — funkcja obliczająca pierwiastek kwadratowy
STRIG — funkcja, która podaje aktualny stan manetki
STICK — funkcja podająca kierunek nachylenia drążka manetki
STOP ON(OFF)STOP — takie samo działanie jak w przypadku SPRITE ON/OFF, ale odnoszące się do klawisza STOP
STR$ — funkcja, która zamienia liczbę na postać tekstową
STRIG ON/OFF/STOP — jw., ale w odniesieniu do przycisku manetki
STRING — ciąg znaków ujętych w cudzysłowy
STRING$ — funkcja pozwalająca tworzyć teksty z wybranych znaków
SWAP — instrukcja zamieniająca miejscami wartości zmiennych
TAB — instrukcja lokująca kursor w zadanym miejscu w wierszu
TAN — funkcja przybierająca wartość równą tangensowi kąta wyrażonego w radianach
TIME — funkcja podająca bieżące wskazania zegara wewnętrznego
TROFF — **TR**ace **OFF** — instrukcja wyłączająca proces śledzenia przebiegu programu
TRON — **TR**ace **ON** — instrukcja, która włącza proces śledzenia, czy program przebiega zgodnie z przewidywaniami
VAL — funkcja, która zamienia tekst na liczbę
VERIFY — rozkaz porównania treści programu zapisanego na taśmie z zawartością pamięci komputera
VPEEK — funkcja, która podaje zawartość wskazanej komórki pamięci ekranu
WIDTH — rozkaz ograniczający liczbę znaków wyświetlanych na ekranie w jednym wierszu.

Literatura

1. Bielecki J.; *Języki programowania mikrokomputerów*. Wyd. Politechniki Warszawskiej, 1987.
2. Czech Z., Nałęcki K., Wołek B.; *Programowanie w języku BASIC*. Wyd. 3, WNT 1986.
3. Freiek B., Lewandowski A.; *Mikrokomputer — programowanie w języku BASIC*. Wyd. NOT Sigma 1986.
4. Iszkowski W., Maniecki M.; *Programowanie w języku BASIC*. Wyd. 3, PWN 1986.
5. Iszkowski W.; *Nauka programowania w języku BASIC dla początkujących*. WNT 1987.
6. Jakubicki H., Pawłowska A.; *Obsługa mikrokomputera ZX Spectrum i programowanie w języku BASIC*. Wyd. Akademia Rolnicza, Wrocław 1986.
7. Jeske A., Stryjski R.; *Podstawy programowania minikomputerów w języku BASIC*. PTE, Zielona Góra 1986.
8. Kalinowska-Iszkowska M., Iszkowski W.; *Klucze do Basicu*. WNT 1987.
9. Kobus A., Szyller J.; *Mikroprocesory*. WP 1989.
10. Kwiatkowska A., Łatko M., Rajtar S.; *Programowanie w języku BASIC dla mikrokomputerów IMZ — 80*. Wyd. Politechniki Lubelskiej 1986.
11. Misiurewicz P.; *Systemy mikrokomputerowe*. Wyd. 2 WSiP 1986.
12. Puch A.; *BASIC — wstęp do programowania i metod numerycznych*. Wyd. Wyższej Szkoły Pedagogicznej, Rzeszów 1986.
13. Sirko A.; *BASIC dla mikrokomputera COBRA 1*. WNT 1986.
14. Tatarkiewicz J., Witkowski A.; *Dwa do piątej numerycznych programów w języku BASIC*. Wyd. NOT Sigma 1987.
15. Wirth N.: *Wstęp do programowania systematycznego*. Przekł. Grzymkowski M. i Prus-Grzymkowska D., WNT 1987.
16. *Język programowania BASIC: wersja na ZX Spectrum*. Wyd. ZETO, Bydgoszcz 1986.
17. [praca zbiorowa]; *Wprowadzenie do programowania w języku BASIC mikrokomputerów MERA 60 oraz w języku FORTRAN 1900 komputerów ODRA 1300*. Wyd. Politechniki Śląskiej, Gliwice 1983.

Spis treści

PW „Wiedza Powszechna", Warszawa 1989 r.
Wydanie I. Nakład 9700+300 egz.
Ark. wyd. 8,6. Ark. druk. 10,75.
Zakłady Graficzne w Katowicach,
Zakład nr 2 w Chorzowie,
ul. Belojannisa 15, 41-500 Chorzów.
Zam. 5018/9 — A-84