COBOL

Autor: Mo Budlong
Tłumaczenie: Grzegorz Kowalczyk
Tytuł oryginału: Teach Yourself COBOL in 21 Days
ISBN:83-7197-534-1
Liczba stron: 658
Nośnik: CD

Książka omawia programowanie w języku COBOL. Język ten jest obecnie rzadko używany, ale zajmuje on poczesne miejsce w historii informatyki. Pomimo iż jest to dzisiaj język trochę egzotyczny dla „przeciętnego" informatyka, to jednak warto się z nim zapoznać, ponieważ do dnia dzisiejszego znajduje on zastosowania w specyficznych dziedzinach np. bankowości (dotyczy to starszych systemów) czy też w kolejnictwie (przynajmniej w Polsce).
Książka jest przeznaczona dla osób, które chcą poszerzyć już posiadaną wiedzę o COBOL- u, bądź też chcą go poznać od podstaw.
Książka została opracowana w taki sposób, aby informacje w niej zawarte można było wykorzystać na większości platform i dla zdecydowanej większości kompilatorów języka COBOL. Jak można się spodziewać, pomiędzy różnymi platformami i wersjami kompilatorów mogą występować dość znaczące różnice w sposobie chociażby metod komunikacji z użytkownikiem.
Wszystkie zaprezentowaneprzykłady powinny bez żadnych trudności działać z takimi wersjami kompilatorów jak Micro Focus COBOL, ACUCOBOL, Fuijtsu COBOL97 oraz IBM Visual Age COBOL dla platform UNIX i DOS a także dla wersji VAX COBOL na platformie VAX VMS. Bez specjalnych kłopotów nasze programy powinny również działać z kompilatorami IBM COBOL na platformie AIX, oraz kompilatorami RM COBOL oraz Realia COBOL.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Rozdział 7.

Podstawy projektowania programów

Zapoznałeś się już z wieloma aspektami programowania w języku COBOL i praktycznie możesz tworzyć swoje własne programy. Do tworzenia programów warto jednak podejść w usystematyzowany sposób. W niniejszym rozdziale zapoznasz się z następującymi zagadnieniami:

- Do czego potrzebne są programy?
- Tworzenie opisu problemu.
- Rozkładanie złożonego problemu na zadania składowe.
- Identyfikacja pętli przetwarzania.
- Czym jest pseudokod i jak z niego korzystać?
- Wyodrębnianie głównej pętli przetwarzania.
- Etapy projektowania programu.
- Program obliczający procent składany.

Do czego są potrzebne programy?

Zasadniczym zadaniem większości programów jest wykonywanie rozbudowanych, powtarzających się zadań, które człowiekowi zajęłyby długie godziny żmudnej, nużącej pracy i zredukowanie problemu do sekwencji dobrze zdefiniowanych poleceń wykonywanych przez maszynę.

Dany problem dobrze nadaje się do rozwiązywania przez program wtedy, kiedy wymaga realizacji pewnych zadań w powtarzalny sposób. Dobrym przykładem może być program zarządzający biblioteką, gdzie należy regularnie przeglądać listę wypożyczonych książek, sprawdzać ich terminy zwrotu, wyszukiwać książki, które powinny już

zostać zwrócone i drukować upomnienia dla nierzetelnych czytelników. Rzeczywiście jest to powtarzalne zadanie, ponieważ każda książka musi zostać sprawdzona według poniższego wzorca:

1. Sprawdź datę wypożyczenia.
2. Sprawdź, czy nie minął termin zwrotu.
3. Jeżeli tak, to wydrukuj upomnienie.

Dany problem można również rozwiązać, stosując omawiany program, wtedy, kiedy operacja jest wykonywana jednorazowo, ale program może być wykorzystywany później wielokrotnie. Przykładowo, taki program może być przydatny do projektowania i drukowania etykiet na potrzeby sklepu. Dopóki nie zależy Ci na tym, aby każda etykieta była na swój sposób niepowtarzalna, taki program może znakomicie spełnić swoje zadanie.

Z drugiej strony, jeżeli planujesz np. okazjonalną wyprzedaż używanych narzędzi, to tworzenie osobnego programu na potrzeby tego wydarzenia nie jest chyba zbyt dobrym pomysłem (chyba, że organizujesz taką wyprzedaż przynajmniej raz na tydzień).

W pierwszych rozdziałach naszej książki przedstawiono kilka programów, które w zasadzie nie były zbyt dobrymi kandydatami na „prawdziwe" programy. Celem przyświecającym utworzeniu takich programów jak `Hello world` była chęć zilustrowania zasad pisania programów w języku COBOL.

Utworzenie programów liczących i wyświetlających tabliczkę mnożenia także było jak najbardziej usprawiedliwione — wykonują one powtarzające się operacje, polegające na wyliczaniu kolejnych iloczynów i wyświetlaniu otrzymanych wyników.

Niestety, niektóre powtarzalne zadania są trudne do implementacji w programach po prostu ze względu na fizyczne ograniczenia komputerów. Przykładowo załóżmy, że codziennie rano robisz sobie filiżankę kawy zawsze w ten sam sposób — jest to jak najbardziej powtarzalne zadanie, aczkolwiek napisanie programu, który by je realizował może być trudne. Ekspres do kawy jest standardowym urządzeniem wejścia-wyjścia póki co w… niewielkiej ilości komputerów.

Jeżeli zadanie wymaga przetwarzania danych, to mogę się założyć, że dobry program może być tutaj bardzo pomocny (choć i to nie zawsze jest prawdą… Wyobraź sobie powtarzalne zadanie polegające na wyborze spośród wszystkich Twoich znajomych 50 osób, które zaprosisz na party. Dopóki komputer nie będzie wiedział, kogo tak naprawdę lubisz, nie poradzi sobie z tym problemem, mimo że jest to problem powtarzalny i wymagający przetwarzania danych…).

Tworzenie opisu problemu

Kiedy już zadecydujesz, czy dane zadanie będzie się dobrze nadawało do rozwiązania przez program, to kolejnym krokiem przybliżającym Cię do sukcesu będzie stworzenie

opisu tego zadania. Opis nie musi być bardzo szczegółowy, ale na pewno musi być precyzyjny. Opisy do niektórych programów prezentowanych w poprzednich rozdziałach mogłyby wyglądać tak:

- Sprawdź wypożyczone książki i wyślij upomnienia do wszystkich czytelników, którzy nie oddali książek w terminie.
- Pobierz treść etykiety podaną przez użytkownika, utwórz etykietę zgodnie z ustalonym wzorcem, a następnie wydrukuj jedną lub kilka kopii.
- Wyświetl wybraną przez użytkownika liczbę wierszy tabliczki mnożenia dla podanej mnożnej z zakresu od 1 do 99.
- Wyświetl wybraną przez użytkownika liczbę wierszy tabliczki mnożenia dla podanej mnożnej z zakresu od 1 do 99, łącząc wyświetlane wiersze w grupy.

Zwróć uwagę na różnicę pomiędzy ostatnimi dwoma zadaniami. Jest to przykład precyzyjnego, ale pozbawionego większych szczegółów opisu zadania — opisy dwóch ostatnich zadań są bardzo do siebie zbliżone, ale jednak inne.

Rozkładanie złożonego problemu na zadania składowe

Po utworzeniu opisu zadania powinieneś przystąpić do rozkładania problemu na prostsze zadania składowe. Na tym etapie projektowania musisz już posiadać pewną wiedzę na temat możliwości języka, w którym będziesz implementował dany program.

Nie przejmując się kolejnością poszczególnych zadań, dokonaj rozbioru problemu na zadania składowe, krótko opisując, co komputer będzie musiał w każdym z nich wykonać.

„Normalny" opis zadania	Opis zadania, które musi wykonać komputer
Wyświetlaj kolejne tabliczki mnożenia	Wyświetlaj w pętli kolejne tabliczki mnożenia
Dla dowolnej mnożnej z zakresu od 1 do 99	Poproś użytkownika o wprowadzenie numeru tabeli (czyli inaczej mnożnej) z zakresu 1 – 99
Wyświetl wybraną ilość kolejnych iloczynów	1. Poproś użytkownika o wprowadzenie ilości wierszy do wyświetlenia 2. Wyświetlaj iloczyny mnożnej i kolejnych mnożników od 1 do liczby podanej przez użytkownika

Takie „małe" zadania, jak te powyżej, mogą z reguły zostać podzielone na jeszcze bardziej elementarne części. Proces rozbijania problemu na coraz mniejsze zadania musi być realizowany dopóty, dopóki wszystkie zadania nie zostaną zapisane w sposób, który będzie bardziej „zrozumiały" dla komputera.

Identyfikacja pętli przetwarzania

Poszczególne pętle przetwarzania zaczynają się wyłaniać w momencie rozbijania złożonego problemu na zadania składowe. Jak zapewne sobie przypominasz, w rozdziale 5., *Zastosowanie poleceń PERFORM, GO TO oraz IF do sterowania przebiegiem działania programu*, pętlę zdefiniowaliśmy jako dowolne zadanie, które jest powtarzane wielokrotnie w jednej sekwencji.

Na obecnym etapie realizacji zadania powinieneś już być w stanie wyodrębnić dwie pętle. Pierwsza z nich wyświetla całą tabelę iloczynów dla podanego mnożnika (numeru tabeli), z kolei druga wyświetla kolejne iloczyny mnożnej i mnożnika, aż do momentu osiągnięcia przez mnożnik wartości zadanej przez użytkownika.

Po zdefiniowaniu pętli warto na nią spojrzeć z nieco innej perspektywy i spróbować ją sobie wyobrazić jako pętlę, która wielokrotnie wykonuje tylko jedno zadanie, a nie kilka zadań naraz. Tabela 7.1 ilustruje różnicę pomiędzy tradycyjnym i „komputerowym" podejściem do zadań wykonywanych przez pętle.

Tabela 7.1. *Myślenie tradycyjne kontra „komputerowe"*

Myślenie tradycyjne	„Komputerowe" podejście do zagadnienia
Wyświetl kilka różnych tabel iloczynów	Wyświetlaj w pętli tabelę iloczynów, za każdym razem zmieniając wartości mnożnej
Wyświetl tabelę kolejnych iloczynów dla podanej mnożnej	Wyświetlaj w pętli kolejne iloczyny dla podanej mnożnej, za każdym razem zmieniając wartości mnożnika

Pseudokod — to wygodna forma zapisu kodu programu, pozwalająca na bardzo swobodne potraktowanie składni poleceń. *Pseudokod* umożliwia również pozostawianie wolnych miejsc w zapisie „programu", np. wtedy, kiedy nie wiesz jeszcze, jak rozwiązać dany fragment zadania. Dobry *pseudokod* powinien być bardzo zbliżony do rzeczywistego języka programowania — w praktyce nie ma żadnych ścisłych reguł określających zasady zapisu „programu" w pseudokodzie, poza jedną: pseudokod powinien Ci pomóc w tworzeniu programu.

Na obecnym etapie posiadasz już ściśle zdefiniowany problem, określone poszczególne zadania składowe, wyznaczone główne pętle przetwarzania; najwyższy więc czas, aby zabrać się za próbę połączenia tych elementów w jednym programie — na razie napisanym w pseudokodzie.

Pierwszym etapem będzie utworzenie głównego zadania całego programu. W tym miejscu musisz zastanowić się, co tak naprawdę program będzie robił? Z punktu widzenia użytkownika można powiedzieć, że będzie wyświetlał tabliczkę mnożenia, jednak myśląc po „komputerowemu", należałoby raczej powiedzieć, że zadaniem programu jest wyświetlanie serii kolejnych iloczynów, których parametry zmieniają się za każdym przebiegiem programu. W listingu 7.1 przedstawiono pseudokod realizujący takie zadanie.

Zwróć uwagę, że pseudokod używany w tym przykładzie nie jest do końca zgodny z zasadami składni języka COBOL — nie wszystkie słowa zostały napisane wielkimi literami, a poszczególne polecenia nie muszą kończyć się kropkami. Jest to typowa cecha pseudokodu — jest jak COBOL, ale stosując go, nie musisz przestrzegać wszystkich zasad składni.

Listing 7.1. *Pierwszy program zapisany w pseudokodzie*

```
THE-PROGRAM
    DISPLAY-ONE-TABLE over and over
```

Określenie *over and over*, czyli oględnie mówiąc, *wykonywanie pewnych operacji „w kółko”*, jest być może wystarczająco dokładne dla programisty, ale komputer wymaga bardziej precyzyjnych poleceń, aby wiedzieć, kiedy zatrzymać działanie. Takie instrukcje przekładają się w prosty sposób na warunek `UNTIL`. Zastosowanie tej klauzuli ilustruje kolejny program przedstawiony w listingu 7.2.

Listing 7.2. *Główne zadanie programu zapisane w pseudokodzie*

```
THE-PROGRAM
    DISPLAY-ONE-TABLE
UNTIL  warunek_zakończenia???
```

W kolejnym etapie przyjrzymy się blokowi `DISPLAY-ONE-TABLE`. Jakie operacje muszą zostać wykonane, aby na ekranie została wyświetlona jedna tabela iloczynów? W uproszczeniu można powiedzieć, że potrzebne będą tylko dwie operacje: pobranie numeru tabeli i wyświetlenie tej tabeli. W listingu 7.3 przedstawiono pseudokod rozszerzony o opisane kroki.

Listing 7.3. *Nowa wersja pseudoprogramu*

```
THE-PROGRAM
   DISPLAY-ONE-TABLE
UNTIL  warunek_zakończenia???

DISPLAY-ONE-TABLE
    GET-WHICH-TABLE
    DISPLAY-THE-TABLE
```

Zapis `GET-THE-TABLE` może zostać w języku COBOL przełożony na proste polecenia, które wyświetlają na ekranie odpowiedni komunikat, a następnie pobierają wartość podaną przez użytkownika. OK, a co należy zrobić, aby wyświetlić tabelę? Tutaj także będziesz musiał wykonać dwie operacje. Po pierwsze, pobierz ilość pozycji do wyświetlenia (czyli maksymalną wartość mnożnika). Następnie wyświetlaj kolejne iloczyny, za każdym razem zwiększając wartość mnożnika dopóty, dopóki nie zostanie osiągnięta maksymalna wartość mnożnika. W listingu 7.4 przedstawiono kolejną wersję programu, z rozbudowanymi blokami `GET-WHICH-TABLE` oraz `DISPLAY-THE-TABLE`.

źródło **Listing 7.4.** *Kolejna wersja pseudoprogramu*

```
THE-PROGRAM
   DISPLAY-ONE-TABLE
UNTIL  warunek_zakończenia???

DISPLAY-ONE-TABLE
    GET-WHICH-TABLE
    DISPLAY-THE-TABLE

GET-WHICH-TABLE
    DISPLAY "Which table? (01-99)"
    ACCEPT THE-TABLE

DISPLAY THE-TABLE
    GET-HOW-MANY-ENTRIES
    DISPLAY-ONE-ENTRY
        UNTIL wszystkie_iloczyny_wyświetlone
```

W tym momencie możesz już zastosować nabytą do tej pory wiedzę o języku COBOL. Program powinien wyświetlać kolejne iloczyny dla mnożników od 1 do wartości podanej przez użytkownika. Jest to znakomita okazja, aby zastosować klauzulę `VARYING`. Pseudoprogram przedstawiony w listingu 7.5 zawiera rozbudowaną wersję bloku `DISPLAY-THE-TABLE` oraz blok umożliwiający użytkownikowi wprowadzenie ilości wierszy do wyświetlenia (czyli maksymalnej wartości mnożnika).

źródło **Listing 7.5.** *Program zaczyna nabierać kształtu…*

```
THE-PROGRAM
   DISPLAY-ONE-TABLE
UNTIL  warunek_zakończenia???

DISPLAY-ONE-TABLE
    GET-WHICH-TABLE
    DISPLAY-THE-TABLE

GET-WHICH-TABLE
    DISPLAY "Which table? (01-99)"
    ACCEPT THE-TABLE

DISPLAY THE-TABLE
    GET-HOW-MANY-ENTRIES
    DISPLAY-ONE-ENTRY
        VARYING THE-ENTRY FROM 1 BY 1
        UNTIL THE-ENTRY > HOW-MANY-ENTRIES

GET-HOW-MANY-ENTRIES
    DISPLAY "How many entries (01-99)?"
    ACCEPT HOW-MANY-ENTRIES
```

Ostatnim etapem jest wyświetlenie pojedynczego wiersza (iloczynu). Listing 7.6 przedstawia niemal zakończony pseudokod programu.

źródło **Listing 7.6.** *Szkielet programu zapisany w pseudokodzie*

```
THE-PROGRAM
   DISPLAY-ONE-TABLE
UNTIL  warunek_zakończenia???

DISPLAY-ONE-TABLE
    GET-WHICH-TABLE
    DISPLAY-THE-TABLE

GET-WHICH-TABLE
    DISPLAY "Which table? (01-99)"
    ACCEPT THE-TABLE

DISPLAY THE-TABLE
    GET-HOW-MANY-ENTRIES
    DISPLAY-ONE-ENTRY
        VARYING THE-ENTRY FROM 1 BY 1
        UNTIL THE-ENTRY > HOW-MANY-ENTRIES

GET-HOW-MANY-ENTRIES
    DISPLAY "How many entries (01-99)?"
    ACCEPT HOW-MANY-ENTRIES

DISPLAY-ONE-ENTRY
    COMPUTE THE-PRODUCT = THE-TABLE * THE-ENTRY
    DISPLAY THE-TABLE " * "
            THE-ENTRY " = "
            THE-PRODUCT
```

Zapewne nie chcesz, aby program był wykonywany w pętli nieskończonej, więc powinieneś jeszcze dopisać warunek zakończenia programu w klauzuli `UNTIL` — powinien on umożliwiać zakończenie działania programu na życzenie użytkownika. Możesz to osiągnąć poprzez zadanie użytkownikowi pytania, czy chce kontynuować działanie programu.

W jakim momencie należy zadać to pytanie? Odpowiedź brzmi: po zakończeniu wyświetlania pełnej tabeli iloczynów dla podanej mnożnej. Jak pamiętasz, dwoma kluczowymi miejscami każdej pętli są jej początek i koniec. Powyższe pytanie może być zadawane na końcu bloku `DISPLAY-ONE-TABLE`. Listing 7.7 przedstawia pseudokod programu uzupełniony o wywołanie bloku `GO-AGAIN` oraz definicję tego bloku. W bloku `GO-AGAIN` mogłem umieścić rozbudowany podprogram sprawdzający np. czy użytkownik wprowadził jedną z odpowiedzi `Y`, `y`, `N`, `n`, a w przypadku udzielenia innej odpowiedzi wyświetlający odpowiedni komunikat — choć w tak prostym programie byłby to chyba już przerost formy nad treścią… Zamiast tego program sprawdza tylko, czy odpowiedź użytkownika brzmi `Y` lub `y` — każda inna odpowiedź jest traktowana jako wprowadzenie odpowiedzi `N`.

źródło **Listing 7.7.** *Prawie ukończona wersja pseudokodu…*

```
THE-PROGRAM
   DISPLAY-ONE-TABLE
UNTIL  YES-NO = "N"

DISPLAY-ONE-TABLE
```

```
    GET-WHICH-TABLE
    DISPLAY-THE-TABLE
    GO-AGAIN

GET-WHICH-TABLE
    DISPLAY "Which table? (01-99)"
    ACCEPT THE-TABLE

DISPLAY THE-TABLE
    GET-HOW-MANY-ENTRIES
    DISPLAY-ONE-ENTRY
        VARYING THE-ENTRY FROM 1 BY 1
        UNTIL THE-ENTRY > HOW-MANY-ENTRIES

GO-AGAIN
    DISPLAY "Go Again (Y/N)?"
    ACCEPT YES-NO
    IF YES-NO = "y"
        MOVE "Y" TO YES-NO
    IF YES-NO NOT = "Y"
        MOVE "N" TO YES-NO

GET-HOW-MANY-ENTRIES
    DISPLAY "How many entries (01-99)?"
    ACCEPT HOW-MANY-ENTRIES

DISPLAY-ONE-ENTRY
    COMPUTE THE-PRODUCT = THE-TABLE * THE-ENTRY
    DISPLAY THE-TABLE " * "
            THE-ENTRY " = "
            THE-PRODUCT
```

Nadszedł wreszcie czas na uporządkowanie i ostateczne uzupełnienie naszego dzieła. Po pierwsze, spójrzmy krytycznym okiem na pętle. Definicja pierwszej pętli w programie może kryć w sobie potencjalny problem. Przypomnijmy sobie, w jaki sposób powinna działać dobrze zaprojektowana pętla:

1. Nadaj zmiennym sterującym działaniem pętli wartości początkowe.
2. Wykonaj pętlę.
3. Zmodyfikuj zmienną sterującą — możesz tego dokonać albo w ostatnim poleceniu pętli, albo po każdym wykonaniu pętli.

Dla pętli realizowanej w bloku `DISPLAY-ONE-TABLE` nie został dopełniony warunek 1 — możesz jednak bardzo łatwo naprawić ten „błąd” poprzez wymuszenie nadania zmiennej `YES-NO` początkowej wartości `"Y"`. Takie postępowanie zapewnia prawidłowe, pierwsze wykonanie tej pętli.

```
THE-PROGRAM
   MOVE "Y" TO YES-NO
   DISPLAY-ONE-TABLE
       UNTIL  YES-NO = "N"
```

Kolejną sprawą, którą musimy się jeszcze zająć jest problem wyświetlania max. 15 wierszy jednocześnie. Możesz go rozwiązać poprzez umieszczenie na początku pętli DISPLAY-ONE-ENTRY poleceń wyświetlających komunikat Press ENTER... i zatrzymujących wyświetlanie, a na końcu pętli — polecenia inkrementacji licznika wyświetlanych wierszy (zmienna SCREEN-LINES).

Ponieważ program może wyświetlać kilka tabel iloczynów po 15 wierszy w każdym, to przed wyświetleniem kolejnych 15 wierszy zmiennej SCREEN-LINES powinna być nadawana wartość 0. W listingu 7.8 przedstawiono finalną wersję pseudokodu programu.

źródło **Listing 7.8.** *Finalna wersja pseudokodu programu*

```
THE-PROGRAM
   MOVE "Y" TO YES-NO
DISPLAY-ONE-TABLE
       UNTIL  YES-NO = "N"

DISPLAY-ONE-TABLE
    GET-WHICH-TABLE
    DISPLAY-THE-TABLE
    GO-AGAIN

GET-WHICH-TABLE
    DISPLAY "Which table? (01-99)"
    ACCEPT THE-TABLE

DISPLAY THE-TABLE
    GET-HOW-MANY-ENTRIES
    MOVE 0 TO SCREEN-LINES
    DISPLAY-ONE-ENTRY
        VARYING THE-ENTRY FROM 1 BY 1
        UNTIL THE-ENTRY > HOW-MANY-ENTRIES

GO-AGAIN
    DISPLAY "Go Again (Y/N)?"
    ACCEPT YES-NO
    IF YES-NO = "y"
        MOVE "Y" TO YES-NO
    IF YES-NO NOT = "Y"
        MOVE "N" TO YES-NO

GET-HOW-MANY-ENTRIES
    DISPLAY "How many entries (01-99)?"
    ACCEPT HOW-MANY-ENTRIES

DISPLAY-ONE-ENTRY
    IF SCREEN-LINES = 15
        PRESS-ENTER
    COMPUTE THE-PRODUCT = THE-TABLE * THE-ENTRY
    DISPLAY THE-TABLE " * "
            THE-ENTRY " = "
            THE-PRODUCT
    ADD 1 TO SCREEN-LINES

PRESS-ENTER
    DISPLAY "Press ENTER to continue"
    ACCEPT A-DUMMY
```

Co teraz? Tworząc powyższy program, starałem się tak dobierać pseudokod, aby można go było łatwo zapisać w języku COBOL — pamiętaj, że nadrzędnym celem stosowania pseudokodu jest pomoc w projektowaniu i tworzeniu docelowego programu. Jeżeli przyjrzysz się listingowi 7.8, to z pewnością zauważysz, że nasz zapis w pseudokodzie jest bardzo zbliżony do rzeczywistej składni języka COBOL.

Prace nad konwersją pseudokodu powinieneś rozpocząć od uzupełniania interpunkcji programu (wstawianie kropek w odpowiednich miejscach). Następnie dodaj niezbędne polecenia `PERFORM`, utwórz sekcję `WORKING-STORAGE` oraz definicje odpowiednich zmiennych i wreszcie skompiluj, uruchom i przetestuj cały program. Istniejący pseudokod jest niemal identyczny jak polecenia, które powinny znaleźć się w obszarze `PROCEDURE DIVISION`. Pamiętaj, że wiele kompilatorów języka COBOL, m.in. ACUCOBOL, wymaga podawania pełnych wartości liczbowych, łącznie z zerami wiodącymi (np. `05` zamiast `5`).

Zwróć uwagę, że podanie ujemnych wartości w odpowiedzi na pytania zadawane przez program może przynieść nieprzewidziane rezultaty. Jako samodzielne ćwiczenie możesz spróbować poprawić kod programu tak, aby poradził sobie z nieprawidłowymi danymi wejściowymi.

źródło **Listing 7.9.** *Pełny program zapisany w języku COBOL*

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. MULT07.
000300*-------------------------------------------------
000400* Program pyta użytkownika
000500* o numer tabliczki mnożenia, rozmiar tabliczki
000600* a następnie wyświetla tę tabliczkę mnożenia
000700* dla wartości od 1 do HOW-MANY.
000800*
000900* Wyświetlanie jest przerywane po każdych kolejnych 15 wierszach.
001000*-------------------------------------------------
001100 ENVIRONMENT DIVISION.
001200 DATA DIVISION.
001300 WORKING-STORAGE SECTION.
001400
001500 01  THE-TABLE          PIC 99.
001600 01  THE-ENTRY          PIC 999.
001700 01  THE-PRODUCT        PIC 9999.
001800 01  HOW-MANY-ENTRIES   PIC 99.
001900 01  SCREEN-LINES       PIC 99.
002000
002100 01  A-DUMMY            PIC X.
002200
002300 01  YES-NO             PIC X.
002400
002500 PROCEDURE DIVISION.
002600
002700 PROGRAM-BEGIN.
002800     MOVE "Y" TO YES-NO.
002900     PERFORM DISPLAY-ONE-TABLE
003000         UNTIL YES-NO = "N".
003100
```

```
003200 PROGRAM-DONE.
003300     STOP RUN.
003400
003500 DISPLAY-ONE-TABLE.
003600     PERFORM GET-WHICH-TABLE.
003700     PERFORM DISPLAY-THE-TABLE.
003800     PERFORM GO-AGAIN.
003900
004000 GET-WHICH-TABLE.
004100     DISPLAY
004200      "Which multiplication table(01-99)?".
004300     ACCEPT THE-TABLE.
004400
004500 DISPLAY-THE-TABLE.
004600     PERFORM GET-HOW-MANY-ENTRIES.
004700
004800     MOVE 0 TO SCREEN-LINES.
004900
005000     PERFORM DISPLAY-ONE-ENTRY
005100         VARYING THE-ENTRY
005200           FROM 1 BY 1
005300           UNTIL THE-ENTRY > HOW-MANY-ENTRIES.
005400
005500 GO-AGAIN.
005600     DISPLAY "Go Again (Y/N)?".
005700     ACCEPT YES-NO.
005800     IF YES-NO = "y"
005900         MOVE "Y" TO YES-NO.
006000     IF YES-NO NOT = "Y"
006100         MOVE "N" TO YES-NO.
006200
006300 GET-HOW-MANY-ENTRIES.
006400     DISPLAY
006500      "How many entries would you like (01-99)?".
006600     ACCEPT HOW-MANY-ENTRIES.
006700
006800 DISPLAY-ONE-ENTRY.
006900
007000     IF SCREEN-LINES = 15
007100         PERFORM PRESS-ENTER.
007200     COMPUTE THE-PRODUCT = THE-TABLE * THE-ENTRY.
007300     DISPLAY
007400         THE-TABLE " * " THE-ENTRY " = " THE-PRODUCT.
007500
007600     ADD 1 TO SCREEN-LINES.
007700
007800 PRESS-ENTER.
007900     DISPLAY "Press ENTER to continue . . .".
008000     ACCEPT A-DUMMY.
008100     MOVE 0 TO SCREEN-LINES.
008200
```

Wyodrębnianie głównej pętli przetwarzania

Główna pętla przetwarzania programu, czyli główne zadanie realizowane przez program, nie zawsze jest zadaniem powtarzanym wielokrotnie. Przykłady takich pętli znajdziesz we wcześniejszych wersjach programu wyświetlającego tabliczkę mnożenia, z którymi spotkałeś się w rozdziale 5. naszej książki.

Powstaje tutaj pytanie: jeżeli główna pętla przetwarzania nie musi tak naprawdę być pętlą, to jak można ją zidentyfikować w programie? Jednym ze sposobów jest założenie, że każde zadanie realizowane przez program będzie wykonywane *wielokrotnie*. Przykładowo, jeżeli oryginalny opis zadania mówił, że program będzie wyświetlał tylko jedną tabliczkę mnożenia, to projektując program, możesz założyć, że będzie on *wielokrotnie* wyświetlał tabliczki mnożenia.

Po zastosowaniu tego prostego triku do opisu głównego zadania programu (czyli w praktyce do głównej pętli przetwarzania) przekształcenie programu do wersji jednopętlowej (czyli z wyodrębnioną główną pętlą przetwarzania) nie powinno już sprawiać większych kłopotów.

Interesującą cechą wszystkich pętli jest fakt, że działają one dla jednej iteracji dokładnie tak samo, jak dla wielu iteracji. W listingu 7.10 przedstawiono program *mult08.cbl*, który jest dokładną kopią programu *mult07.cbl*, z tym tylko, że niektóre wiersze oryginalnego programu zostały oznaczone jako komentarz. Jest to bardzo często stosowana praktyka — zamiast usuwania całego wiersza programu wystarczy umieścić w kolumnie 7 znak *, co powoduje, że cały wiersz jest traktowany jako komentarz i pomijany przez kompilator.

Daje to taki sam efekt, jak usunięcie wiersza, ale pozwala na pozostawienie go w kodzie źródłowym. Zazwyczaj takie rozwiązanie stosuje się w sytuacji, kiedy kod programu jest modyfikowany, ale istnieje potrzeba odwołania się do oryginalnego kodu programu.

Listing 7.10. *Konwersja programu mult08.cbl do wersji jednopętlowej*

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. MULT08.
000300*-------------------------------------------------
000400* Program pyta użytkownika
000500* o numer tabliczki mnożenia, rozmiar tabliczki
000600* a następnie wyświetla tę tabliczkę mnożenia
000700* dla wartości od 1 do HOW-MANY.
000800*
000900* Wyświetlanie jest przerywane po każdych kolejnych 15 wierszach.
001000*-------------------------------------------------
001100 ENVIRONMENT DIVISION.
001200 DATA DIVISION.
001300 WORKING-STORAGE SECTION.
001400
001500 01  THE-TABLE          PIC 99.
001600 01  THE-ENTRY          PIC 999.
001700 01  THE-PRODUCT        PIC 9999.
```

```
001800 01  HOW-MANY-ENTRIES   PIC 99.
001900 01  SCREEN-LINES       PIC 99.
002000
002100 01  A-DUMMY            PIC X.
002200
002300*01  YES-NO              PIC X VALUE "Y".
002400
002500 PROCEDURE DIVISION.
002600
002700 PROGRAM-BEGIN.
002800*    MOVE "Y" TO YES-NO.
002900     PERFORM DISPLAY-ONE-TABLE.
003000*        UNTIL YES-NO = "N".
003100
003200 PROGRAM-DONE.
003300     STOP RUN.
003400
003500 DISPLAY-ONE-TABLE.
003600     PERFORM GET-WHICH-TABLE.
003700     PERFORM DISPLAY-THE-TABLE.
003800*    PERFORM GO-AGAIN.
003900
004000 GET-WHICH-TABLE.
004100     DISPLAY
004200      "Which multiplication table(01-99)?".
004300     ACCEPT THE-TABLE.
004400
004500 DISPLAY-THE-TABLE.
004600     PERFORM GET-HOW-MANY-ENTRIES.
004700
004800     MOVE 0 TO SCREEN-LINES.
004900
005000     PERFORM DISPLAY-ONE-ENTRY
005100         VARYING THE-ENTRY
005200           FROM 1 BY 1
005300           UNTIL THE-ENTRY > HOW-MANY-ENTRIES.
005400
005500*GO-AGAIN.
005600*    DISPLAY "Go Again (Y/N)?".
005700*    ACCEPT YES-NO.
005800*    IF YES-NO = "y"
005900*        MOVE "Y" TO YES-NO.
006000*    IF YES-NO NOT = "Y"
006100*        MOVE "N" TO YES-NO.
006200
006300 GET-HOW-MANY-ENTRIES.
006400     DISPLAY
006500      "How many entries would you like (01-99)?".
006600     ACCEPT HOW-MANY-ENTRIES.
006700
006800 DISPLAY-ONE-ENTRY.
006900
007000     IF SCREEN-LINES = 15
007100         PERFORM PRESS-ENTER.
007200     COMPUTE THE-PRODUCT = THE-TABLE * THE-ENTRY.
007300     DISPLAY
007400         THE-TABLE " * " THE-ENTRY " = " THE-PRODUCT.
007500
007600     ADD 1 TO SCREEN-LINES.
```

```
007700
007800 PRESS-ENTER.
007900     DISPLAY "Press ENTER to continue . . .".
008000     ACCEPT A-DUMMY.
008100     MOVE 0 TO SCREEN-LINES.
008200
```

Modyfikacje wprowadzone w listingu 7.10 usuwają cały kod powodujący, że program zadaje użytkownikowi pytanie, czy ponownie wykonać program i powtarzający go, jeżeli użytkownik da odpowiedź twierdzącą.

Z programu oryginalnego zostały usunięte (oznaczone jako komentarz) wiersze o numerach 002300, 002800, 003000, 003800 oraz od 005500 do 006100. Powstały w wyniku tej operacji program różni się od oryginału tym, że tylko jednokrotnie wyświetla tabliczkę mnożenia dla zadanej mnożnej. Program *mult07.cbl* został więc „przykrojony" tak, że zostaje wykonany tylko raz — a dokonano tego wyłącznie przez przemyślane umieszczenie w kodzie źródłowym kilku gwiazdek *.

Wyodrębnienie pętli przetwarzania nie zawsze jest takie oczywiste. Oryginalne wersje poprzednich programów wyświetlających tabliczkę mnożenia (*mult01.cbl* do *mult06.cbl*), z którymi pracowałeś w rozdziale 5., również wykonywały tylko jeden przebieg głównej pętli przetwarzania.

Jeżeli projektujesz program, który wykonuje tylko jeden przebieg danego zadania, to możesz zasymulować proces zilustrowany w listingu 7.10, *mult08.cbl*. Powinieneś wtedy założyć, że program wykonuje wiele przebiegów, zaprojektować cały program, a następnie tak „obciąć" kod programu za pomocą komentarzy, aby główna pętla przetwarzania wykonywana była tylko jeden raz. Jest to prosty trik pozwalający na wyobrażenie sobie zadania realizowanego przez program jako pętli, nawet wtedy, kiedy jest ono wykonywane jeden, jedyny raz.

Podsumowanie etapów tworzenia programu

Zanim rozpoczniemy realizację następnego projektu przypomnijmy sobie, jaka powinna być właściwa kolejność kroków prowadzących do powstania programu (poniższa lista obejmuje etapy zarówno projektowania, jak i implementacji):

1. Utwórz opis zadania. Powinien on precyzyjne, ale niekoniecznie bardzo szczegółowo formułować przeznaczenie programu.
2. Podziel główne zadanie na mniejsze zadania składowe. W tym miejscu musisz dołożyć nieco szczegółowych instrukcji dotyczących wykonania, a sam podział na zadania składowe powinien być prowadzony tak długo, aż poszczególne zadania będą mogły być w prosty sposób zrealizowane przez komputer (program).

3. Wyodrębnij poszczególne pętle przetwarzania.
4. Wyodrębnij główną pętlę przetwarzania (jeżeli nie udało Ci się tego dokonać w punkcie 3).
5. Zapisz program, wykorzystując polecenia pseudokodu.
6. Dokonaj konwersji pseudokodu do postaci prawidłowego kodu źródłowego.
7. Zapisz kod źródłowy, a następnie skompiluj program.
8. Uruchom i przetestuj program.
9. Popraw ewentualne błędy poprzez dokonanie niezbędnych modyfikacji w kodzie źródłowym.
10. Powtarzaj kroki 8. i 9. dopóty, dopóki program nie będzie działał poprawnie.

To normalne, że realizacja punktów od 1. do 5. zajmuje więcej czasu niż punktów od 6. do 10., zwłaszcza, jeżeli bardzo gruntownie i rzetelnie zajmiesz się fazą samego projektowania programu.

Program obliczający procent składany

W niniejszym podrozdziale zajmiemy się projektowaniem zupełnie nowego programu, którego zadaniem będzie obliczanie wielkości lokaty po dodaniu odsetek składanych od kapitału początkowego w danym okresie.

Aby obliczyć procent składany, musisz znać wielkość kapitału początkowego, wysokość oprocentowania wkładu w danym czasie oraz ilość okresów kapitalizacji, przez które odsetki będą kumulowane. Mając na uwadze wytyczne z poprzedniego podrozdziału, rozpoczniemy od utworzenia precyzyjnego opisu zadania:

> *Oblicz wartość lokaty w oparciu o wielkość kapitału początkowego, wysokość oprocentowania oraz ilość okresów kapitalizacji.*

Powyższe zadanie może zostać rozbite na następujące zadania składowe (bez zwracania uwagi na kolejność realizacji zadań):

1. Oblicz wartość lokaty.
2. Wyświetl wartość lokaty.
3. Pobierz wielkość kapitału początkowego.
4. Pobierz wysokość oprocentowania.
5. Pobierz ilość okresów kapitalizacji.

Najprostszą metodą obliczenia procentu złożonego jest obliczenie wartości lokaty po jednym okresie kapitalizacji. Następnie należy wartość tę przyjąć jako nową wartość początkową lokaty i obliczyć kolejną wartość lokaty po jednym okresie kapitalizacji (patrz tabela 7.2). Proces ten powinien być powtarzany aż do obliczenia wartości lokaty

po upływie zadanej ilości okresów kapitalizacji. Tabela 7.2 przedstawia wyliczenie wartości lokaty przy założeniu, że początkowa wartość lokaty wynosi 1000, wysokość oprocentowania wynosi 10%, a ilość okresów kapitalizacji wynosi 4.

Tabela 7.2. *Obliczanie procentu składanego*

Okres	1	2	3	4
Wartość lokaty	1000.00	1100.00	1210.00	1331.00
Wysokość oprocentowania (10%)	x .10	x .10	x .10	x .10
Wysokość odsetek	=100.00	=110.00	=121.00	=133.10
Wartość lokaty + odsetki	+1000.00	+1100.00	+1210.00	+1331.00
Razem	=1100.00	=1210.00	=1331.00	=1464.10

Jak łatwo zauważyć, wartość obliczona na końcu każdego okresu jest przyjmowana jako wartość początkowa kolejnego okresu i cały proces obliczeń jest powtarzany.

Oczywiście, do obliczania wysokości procentu składanego istnieją o wiele bardziej wydajne algorytmy, ale ten właśnie przykład znakomicie ilustruje sytuację, w której nawet nie znając *najlepszego rozwiązania*, możesz wykorzystać komputer do wykonania zadania. Powyższa tabela umożliwia nam dodanie do listy zadań składowych punktu numer 6:

6. Obliczaj nową wartość lokaty po upływie jednego okresu kapitalizacji.

Na podstawie listy zadań składowych można wyodrębnić dwie pętle przetwarzania, które będą odpowiednio realizowały zadania z punktów 1 i 6. Pętla realizująca zadanie numer 1 będzie główną pętlą przetwarzania programu.

Korzystając z nabytej znajomości zasad posługiwania się pseudokodem, utwórzmy szkielet naszego programu. Efekt został przedstawiony w listingu 7.11 — starałem się zastosować w nim formalny pseudokod, który jest bardzo zbliżony do poprawnej składni poleceń języka COBOL. Masz już duże doświadczenie w posługiwaniu się poleceniami typu DISPLAY czy ACCEPT, więc nie ma potrzeby umieszczania tych poleceń w pełnym brzmieniu w pseudokodzie — pamiętaj, że zadaniem pseudokodu jest *pomoc* w tworzeniu programu, a nie przysparzanie programiście dodatkowej pracy. Użyte polecenia pseudokodu zostały sformułowane wystarczająco przejrzyście, aby wskazać, w których miejscach należy wyświetlić komunikaty czy pobrać odpowiednie wartości.

Listing 7.11. *Pseudokod programu obliczającego procenty składane*

```
THE-PROGRAM
    MOVE "Y" TO YES-NO.
PERFORM GET-AND-DISPLAY-RESULT
        UNTIL YES-NO = "N".

GET-AND-DISPLAY-RESULT.
    PERFORM GET-THE-PRINCIPAL.
    PERFORM GET-THE-INTEREST.
    PERFORM GET-THE-PERIODS.
    PERFORM CALCULATE-THE-RESULT.
    PERFORM DISPLAY-THE-RESULT.
```

```
    PERFORM GO-AGAIN.

GET-THE-PRINCIPAL.
    (between 0.01 and 999999.99)

GET-THE-INTEREST.
    (between 0.01 and 99.9%)

GET-THE-PERIODS.
    (between 001 and 999)

CALCULATE-THE-RESULT.
    PERFORM CALCULATE-ONE-PERIOD
        VARYING THE-PERIOD FROM 1 BY 1
            UNTIL THE-PERIOD > NO-OF-PERIODS.

CALCULATE-ONE-PERIOD.
    COMPUTE EARNED-INTEREST ROUNDED =
        THE-PRINCIPAL * INTEREST-AS-DECIMAL.
    COMPUTE THE-NEW-VALUE =
        THE-PRINCIPAL + EARNED-INTEREST.
    MOVE THE-NEW-VALUE TO THE-PRINCIPAL.

GO-AGAIN.
    (YES OR NO)

DISPLAY-THE-RESULT
    (VALUE = THE-PRINCIPAL)
```

W listingu 7.12 przedstawiono kod programu, który powstał na bazie pseudokodu z listingu 7.11. Bloki programu odpowiedzialne za pobieranie wartości początkowej lokaty, wysokości oprocentowania oraz ilości okresów kapitalizacji zostały zaprojektowane tak, aby kontrolowały wprowadzane dane i w razie stwierdzenia nieprawidłowości wyświetlały odpowiednie komunikaty o błędzie i pytały o daną wartość ponownie.

Przeanalizuj przedstawiony w listingu kod źródłowy, a następnie wpisz, skompiluj, uruchom i wypróbuj ten program.

Pamiętaj, że wersja polecenia ACCEPT użyta w programie (np. ACCEPT *zmienna* CONVERT czy ACCEPT *zmienna* WITH CONVERSION) powinna być dostosowana do wersji kompilatora języka COBOL, którą się posługujesz.

Listing 7.12. *Obliczanie procentu składanego*

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. CMPINT01.
000300*-----------------------------------------------
000400* Obliczanie procentu składanego
000500*-----------------------------------------------
000600 ENVIRONMENT DIVISION.
000700 DATA DIVISION.
000800 WORKING-STORAGE SECTION.
000900
```

```
001000 01  YES-NO                          PIC X.
001100 01  THE-INTEREST                    PIC 99V9.
001200 01  INTEREST-AS-DECIMAL             PIC V999.
001300 01  THE-PRINCIPAL                   PIC 9(9)V99.
001400 01  THE-NEW-VALUE                   PIC 9(9)V99.
001500 01  EARNED-INTEREST                 PIC 9(9)V99.
001600 01  THE-PERIOD                      PIC 9999.
001700 01  NO-OF-PERIODS                   PIC 999.
001800
001900 01  ENTRY-FIELD                     PIC Z(9).ZZ.
002000 01  DISPLAY-VALUE                   PIC ZZZ,ZZZ,ZZ9.99.
002100
002200 PROCEDURE DIVISION.
002300 PROGRAM-BEGIN.
002400
002500     MOVE "Y" TO YES-NO.
002600     PERFORM GET-AND-DISPLAY-RESULT
002700         UNTIL YES-NO = "N".
002800
002900 PROGRAM-DONE.
003000     STOP RUN.
003100
003200 GET-AND-DISPLAY-RESULT.
003300     PERFORM GET-THE-PRINCIPAL.
003400     PERFORM GET-THE-INTEREST.
003500     PERFORM GET-THE-PERIODS.
003600     PERFORM CALCULATE-THE-RESULT.
003700     PERFORM DISPLAY-THE-RESULT.
003800     PERFORM GO-AGAIN.
003900
004000 GET-THE-PRINCIPAL.
004100     DISPLAY "Principal (.01 TO 999999.99)?".
004200     ACCEPT ENTRY-FIELD WITH CONVERSION.
004300     MOVE ENTRY-FIELD TO THE-PRINCIPAL.
004400     IF THE-PRINCIPAL < .01 OR
004500        THE-PRINCIPAL > 999999.99
004600         DISPLAY "INVALID ENTRY"
004700         GO TO GET-THE-PRINCIPAL.
004800
004900 GET-THE-INTEREST.
005000     DISPLAY "Interest (.1% TO 99.9%)?".
005100     ACCEPT ENTRY-FIELD WITH CONVERSION.
005200     MOVE ENTRY-FIELD TO THE-INTEREST.
005300     IF THE-INTEREST < .1 OR
005400        THE-INTEREST > 99.9
005500         DISPLAY "INVALID ENTRY"
005600         GO TO GET-THE-INTEREST
005700     ELSE
005800         COMPUTE INTEREST-AS-DECIMAL =
005900                 THE-INTEREST / 100.
006000
006100 GET-THE-PERIODS.
006200     DISPLAY "Number of periods (1 TO 999)?".
006300     ACCEPT ENTRY-FIELD WITH CONVERSION.
006400     MOVE ENTRY-FIELD TO NO-OF-PERIODS.
006500     IF NO-OF-PERIODS < 1 OR
006600        NO-OF-PERIODS > 999
```

```
006700         DISPLAY "INVALID ENTRY"
006800         GO TO GET-THE-PERIODS.
006900
007000 CALCULATE-THE-RESULT.
007100     PERFORM CALCULATE-ONE-PERIOD
007200         VARYING THE-PERIOD FROM 1 BY 1
007300          UNTIL THE-PERIOD > NO-OF-PERIODS.
007400
007500 CALCULATE-ONE-PERIOD.
007600     COMPUTE EARNED-INTEREST ROUNDED =
007700         THE-PRINCIPAL * INTEREST-AS-DECIMAL.
007800     COMPUTE THE-NEW-VALUE =
007900             THE-PRINCIPAL + EARNED-INTEREST.
008000     MOVE THE-NEW-VALUE TO THE-PRINCIPAL.
008100
008200 GO-AGAIN.
008300     DISPLAY "GO AGAIN?".
008400     ACCEPT YES-NO.
008500     IF YES-NO = "y"
008600         MOVE "Y" TO YES-NO.
008700     IF YES-NO NOT = "Y"
008800         MOVE "N" TO YES-NO.
008900
009000 DISPLAY-THE-RESULT.
009100     MOVE THE-NEW-VALUE TO DISPLAY-VALUE.
009200     DISPLAY "RESULTING VALUE IS " DISPLAY-VALUE.
009300
```

Wyniki działania programu

Poniżej przedstawiono wyniki działania programu dla następujących założeń:

- wartość początkowa lokaty: $1,000.00
- oprocentowanie: 1.1% miesięcznie
- ilość okresów kapitalizacji: 48 miesięcy


```
Principal (.01 TO 999999.99)?
1000
Interest (.1% TO 99.9%)?
1.1
Number of periods (1 TO 999)?
48
RESULTING VALUE IS       1,690.65
GO AGAIN?
```

Zanim ktokolwiek zacznie narzekać, że w programie *cmpint01.cbl*, przedstawionym w listingu 7.12, użyto kilku poleceń `GO TO`, przedstawimy program *cmpint02.cbl* (listing 7.13), w którym poprzez zastosowanie poleceń `PERFORM` i zmiennej `ENTRY-OK` uniknięto używania poleceń `GO TO` (przeanalizuj różnice w kodzie między tymi programami). Drugi program jest dobitnym przykładem, że jednak można uniknąć stosowania polecenia `GO TO` nawet wtedy, kiedy jego użycie wydaje się być uzasadnione.

źródło **Listing 7.13.** *Obliczanie procentu składanego (wersja bez polecenia GO TO)*

```
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. CMPINT02.
000300*-----------------------------------------------
000400* Obliczanie procentu składanego
000500*-----------------------------------------------
000600 ENVIRONMENT DIVISION.
000700 DATA DIVISION.
000800 WORKING-STORAGE SECTION.
000900
001000 01  YES-NO                          PIC X.
001100 01  ENTRY-OK                        PIC X.
001200 01  THE-INTEREST                    PIC 99V9.
001300 01  INTEREST-AS-DECIMAL             PIC V999.
001400 01  THE-PRINCIPAL                   PIC 9(9)V99.
001500 01  THE-NEW-VALUE                   PIC 9(9)V99.
001600 01  EARNED-INTEREST                 PIC 9(9)V99.
001700 01  THE-PERIOD                      PIC 9999.
001800 01  NO-OF-PERIODS                   PIC 999.
001900
002000 01  ENTRY-FIELD                     PIC Z(9).ZZ.
002100 01  DISPLAY-VALUE                   PIC ZZZ,ZZZ,ZZ9.99.
002200
002300 PROCEDURE DIVISION.
002400 PROGRAM-BEGIN.
002500
002600     MOVE "Y" TO YES-NO.
002700     PERFORM GET-AND-DISPLAY-RESULT
002800         UNTIL YES-NO = "N".
002900
003000 PROGRAM-DONE.
003100     STOP RUN.
003200
003300 GET-AND-DISPLAY-RESULT.
003400     PERFORM GET-THE-PRINCIPAL.
003500     PERFORM GET-THE-INTEREST.
003600     PERFORM GET-THE-PERIODS.
003700     PERFORM CALCULATE-THE-RESULT.
003800     PERFORM DISPLAY-THE-RESULT.
003900     PERFORM GO-AGAIN.
004000
004100 GET-THE-PRINCIPAL.
004200     MOVE "N" TO ENTRY-OK.
004300     PERFORM ENTER-THE-PRINCIPAL
004400         UNTIL ENTRY-OK = "Y".
004500
004600 ENTER-THE-PRINCIPAL.
004700     DISPLAY "Principal (.01 TO 999999.99)?".
004800     ACCEPT ENTRY-FIELD WITH CONVERSION.
004900     MOVE ENTRY-FIELD TO THE-PRINCIPAL.
005000     IF THE-PRINCIPAL < .01 OR
005100        THE-PRINCIPAL > 999999.99
005200         DISPLAY "INVALID ENTRY"
005300     ELSE
005400         MOVE "Y" TO ENTRY-OK.
005500
```

```
005600 GET-THE-INTEREST.
005700     MOVE "N" TO ENTRY-OK.
005800     PERFORM ENTER-THE-INTEREST
005900         UNTIL ENTRY-OK = "Y".
006000
006100 ENTER-THE-INTEREST.
006200     DISPLAY "Interest (.1% TO 99.9%)?".
006300     ACCEPT ENTRY-FIELD WITH CONVERSION.
006400     MOVE ENTRY-FIELD TO THE-INTEREST.
006500     IF THE-INTEREST < .1 OR
006600        THE-INTEREST > 99.9
006700         DISPLAY "INVALID ENTRY"
006800     ELSE
006900         MOVE "Y" TO ENTRY-OK
007000         COMPUTE INTEREST-AS-DECIMAL =
007100                 THE-INTEREST / 100.
007200
007300 GET-THE-PERIODS.
007400     MOVE "N" TO ENTRY-OK.
007500     PERFORM ENTER-THE-PERIODS
007600         UNTIL ENTRY-OK = "Y".
007700
007800 ENTER-THE-PERIODS.
007900     DISPLAY "Number of periods (1 TO 999)?".
008000     ACCEPT ENTRY-FIELD WITH CONVERSION.
008100     MOVE ENTRY-FIELD TO NO-OF-PERIODS.
008200     IF NO-OF-PERIODS < 1 OR
008300        NO-OF-PERIODS > 999
008400         DISPLAY "INVALID ENTRY"
008500     ELSE
008600         MOVE "Y" TO ENTRY-OK.
008700
008800 CALCULATE-THE-RESULT.
008900     PERFORM CALCULATE-ONE-PERIOD
009000         VARYING THE-PERIOD FROM 1 BY 1
009100          UNTIL THE-PERIOD > NO-OF-PERIODS.
009200
009300 CALCULATE-ONE-PERIOD.
009400     COMPUTE EARNED-INTEREST ROUNDED =
009500         THE-PRINCIPAL * INTEREST-AS-DECIMAL.
009600     COMPUTE THE-NEW-VALUE =
009700             THE-PRINCIPAL + EARNED-INTEREST.
009800     MOVE THE-NEW-VALUE TO THE-PRINCIPAL.
009900
010000 GO-AGAIN.
010100     DISPLAY "GO AGAIN?".
010200     ACCEPT YES-NO.
010300     IF YES-NO = "y"
010400         MOVE "Y" TO YES-NO.
010500     IF YES-NO NOT = "Y"
010600         MOVE "N" TO YES-NO.
010700
010800 DISPLAY-THE-RESULT.
010900     MOVE THE-NEW-VALUE TO DISPLAY-VALUE.
011000     DISPLAY "RESULTING VALUE IS " DISPLAY-VALUE.
011100
```

W przedstawionym programie wprowadzanie danych wejściowych jest zrealizowane w postaci pętli, które są wykonywane dopóty, dopóki nie zostaną wprowadzone poprawne dane.

Przykładowo, w wierszu 004200, zmiennej ENTRY-OK przed wykonaniem pętli ENTER-THE-PRINCIPAL jest nadawana wartość "N". Następnie pętla ENTER-THE-PRINCIPAL jest wykonywana tak długo (klauzula UNTIL), aż zmienna ENTRY-OK otrzyma wartość "Y" (czyli do czasu, aż zostaną wprowadzone poprawne dane wejściowe) — takie rozwiązanie wymusza co najmniej jednokrotne wykonanie tej pętli. Użytkownik zostaje poproszony o wprowadzenie wartości lokaty, wprowadzona wartość zostaje przypisana do odpowiedniej zmiennej, następnie sprawdzona i jeżeli jest poprawna, to program powoduje nadanie zmiennej ENTRY-OK wartości "Y", co kończy działanie pętli (wiersze 004300 i 004400).

Jeżeli wprowadzona wartość nie jest poprawna, to na ekranie jest wyświetlany komunikat INVALID ENTRY, a wartość zmiennej ENTRY-OK nie jest modyfikowana. Powoduje to, że warunek zakończenia pętli PERFORM UNTIL (wiersze 004300 i 004400) nie zostaje spełniony i pętla ENTER-THE-PRINCIPAL jest wykonywana ponownie. Proces ten powtarza się dopóty, dopóki użytkownik nie wprowadzi poprawnych danych i zmiennej ENTRY-OK nie zostanie nadana wartość "Y".