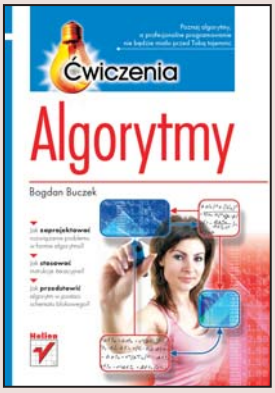
Algorytmy. Ćwiczenia

Autor: Bogdan Buczek
ISBN: 978-83-246-2007-4
Format: A5, stron: 272



Poznaj algorytmy, a profesjonalne programowanie nie będzie miało przed Tobą tajemnic

- Jak zaprojektować rozwiązanie problemu w formie algorytmu?
- Jak stosować instrukcje iteracyjne?
- Jak przedstawić algorytm w postaci schematu blokowego?

W czasach ery informatycznej coraz większa liczba osób zainteresowana jest zdobyciem umiejętności programowania. Jednakże umiejętność ta wymaga zarówno rozległej i rzetelnej wiedzy, jak i doświadczenia. Podstawą owej wiedzy jest dobra znajomość algorytmów, która umożliwia przeprowadzanie kolejnych etapów programowania. Pozwala ona na przechodzenie od analizy i zdefiniowania problemu, poprzez testowanie i usuwanie błędów, aż do opracowania dokumentacji. Książka, którą trzymasz w rękach, pomoże Ci zrozumieć każdą z tych faz i nauczy Cię pisać własny kod.

„Algorytmy. Ćwiczenia" to niezbędny elementarz dla każdego przyszłego programisty. Dzięki temu podręcznikowi poznasz różne sposoby opisu algorytmów oraz ich klasyfikację. Dowiesz się, jaki wpływ ma zastosowanie określonej metody obliczeniowej na dokładność wyników końcowych, a także, na czym polega przetwarzanie danych w pętli programowej. Wykonując kolejne ćwiczenia, opatrzone szczegółowymi komentarzami i wskazówkami, nauczysz się pisać algorytmy, sporządzać wykresy i schematy blokowe oraz tworzyć kod programu. Książka jest doskonałym podręcznikiem dla studentów informatyki, jednak dzięki temu, że wszystkie informacje przedstawiono tu w jasny i klarowny sposób, może z niej korzystać każdy, kto chce rozpocząć samodzielne programowanie.

- Sposoby opisu algorytmów
- Klasyfikacja algorytmów
- Algorytmy sekwencyjne
- Kodowanie algorytmów
- Algorytmy z rozgałęzieniami
- Przetwarzanie danych w pętli programowej
- Algorytmy iteracyjne
- Funkcja silnia
- Instrukcje iteracyjne w Turbo Pascal i Visual Basic
- Algorytmy rekurencyjne
- Schemat Kornera
- Pozycyjne systemy liczbowe
- Algorytmy sortowania danych

Poznaj algorytmy i zacznij myśleć jak programista!

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Algorytmy rekurencyjne

Algorytm rekurencyjny

Rekurencja, zwana również **rekursją**, jest techniką programowania, w której stosowany jest podprogram (funkcja lub procedura) wywołujący sam siebie albo wywołujący inną procedurę, która wywoła podprogram pierwotny. W tym drugim przypadku mówimy o **rekursji podwójnej** lub **skrośnej**. Kolejne wywołania trwają, aż do osiągnięcia warunku zakończenia rekurencji. Jest nim oczekiwany wynik albo przekroczenie rozmiaru zbioru, na którym wykonywane są obliczenia.

Liczba kolejnych wywołań **rekursywnych** nie ma znaczenia. Często jest wręcz niemożliwa do określenia przed rozpoczęciem przetwarzania danych, nie zawsze bowiem da się określić poziom zagłębienia w wywołania.

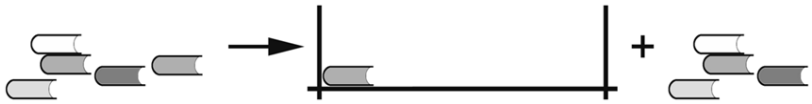
Wynik aktualnie realizowanego obliczenia rekurencyjnego zależy od poprzedzającego go powtórzenia. Każde kolejne wywołanie powoduje zmniejszenie rozmiaru badanego zbioru (np. tablicy) o 1, dzięki czemu problem zostaje rozbity na części elementarne, które operują na mniejszej liczbie danych — są zatem mniej skomplikowane. Dopiero w momencie powrotu z wywołań wyznaczane są wszystkie poprzednie wartości.

Rekurencja wokół nas

Postępowanie o charakterze rekurencyjnym trwale związane jest z wieloma czynnościami zachodzącymi w otaczającej nas rzeczywistości, choć często nie zauważamy tego lub nie jesteśmy świadomi.

Można wskazać wiele przykładów czynności, które mają cechy rekursji, a są wykonywane przez człowieka, zwierzęta albo zaprogramowane automaty. Chodzenie i bieganie, tańczenie, jedzenie, masowe toczenie na tokarce, zbieranie rozsypanych przedmiotów, mycie, zrywanie owoców z drzewa itp.

Równie często opisujemy słownie procesy, stosując język typowy dla rekursji. Instruując kogoś, jak należy myć stos talerzy, mówimy: „Umyj talerz do czysta i myj dalej". Tłumacząc, jak ułożyć na półce rozsypane na podłodze książki, powiemy: „Podnieś książkę, ustaw na półce i podobnie układaj kolejne". Ten schemat postępowania jest przedstawiony graficznie na rysunku 5.1. W obu przykładach czynność jest powtarzana. Różne są jednak warunki zakończenia rekurencji. W pierwszym przykładzie koniec powinien nastąpić, gdy talerze są czyste, w drugim — gdy braknie książek do ustawiania.



Rysunek 5.1. *Model rekurencyjnego układania książek na półce*

Funkcja silnia

Zgodnie z obietnicą daną w poprzednim rozdziale wracamy do funkcji *silnia*. Tym razem poznamy algorytm i rekurencyjne wersje programów wykonujących stosowne obliczenia.

ĆWICZENIE

5.1 Algorytm rekurencyjnego obliczania n!

Przedstaw w postaci schematu blokowego rekurencyjny algorytm obliczania silni $n!$, $n \in N$. Dokonaj analizy przepływu w algorytmie dla $n = 3$.

Rozwiązanie

Dane: Liczba naturalna n wprowadzona przez użytkownika, równa ostatniemu wyrazowi iloczynu.

Oczekiwany wynik: Wartość funkcji $n!$.

Analiza problemu: Definicja *silni* $n!$ liczby naturalnej n wystąpiła w poprzednim rozdziale w ćwiczeniu 4.4. Z definicji klasycznej $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$ wynika własność silni $n! = n(n-1)!$, która pozwala określić tę funkcję w postaci rekurencyjnej:

$$\begin{cases} 0! = 1 \\ n! = n \cdot (n-1)! \end{cases}$$

Obliczenie kolejnej wartości $n!$ następuje poprzez pomnożenie wartości poprzedniej $(n-1)!$ przez następną liczbę naturalną n. Tak zdefiniowana rekurencja nazywana jest *liniową*.

Proces obliczeniowy powinien być powtarzany, aż n osiągnie wartość zadaną przez użytkownika. Na podstawie powyższego można zapisać w innej formie rekurencyjną definicję funkcji silnia:

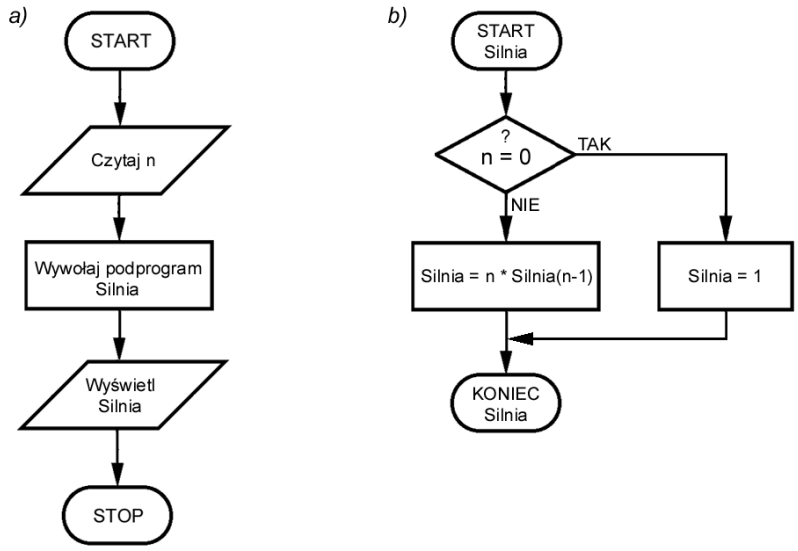
$$\begin{cases} a_0 = 1 \\ a_n = a_{n-1} \cdot n, \qquad n \in N \end{cases}$$

Algorytm przedstawiony na rysunku 5.2 składa się z dwóch części: algorytmu (programu) głównego i podprogramu realizującego rekurencyjne obliczanie funkcji silnia.

Powyższy algorytm można próbować scalić, co pokazuje rysunek 5.3. W tej formie rekurencyjny algorytm obliczania silni występuje w literaturze najczęściej. Niestety obarczony jest poważnym błędem, jakim jest wczytywanie wartości n przy każdym kolejnym odwołaniu rekurencyjnym! Ten algorytm nie działa prawidłowo.

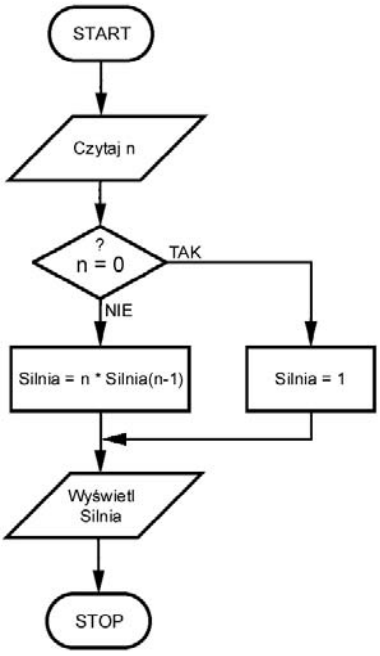
Analiza przepływu w rekurencyjnym algorytmie obliczania silni

W algorytmie z rysunku 5.2 stosowane są dwie zmienne: n — liczba naturalna wprowadzona przez użytkownika (dana wsadowa), *Silnia* — wartość funkcji silnia. Zapis z użyciem nawiasu: *Silnia(argument)* oznacza wartość funkcji dla podanego argumentu, na przykład *Silnia(2)* oznacza wartość funkcji silnia dla $n = 2$.



Rysunek 5.2. *Rekurencyjny algorytm obliczania silni: a) program główny, b) podprogram rekurencyjnego obliczania silni*

Rysunek 5.3. *Błędny algorytm obliczania silni bez użycia podprogramu*



Algorytm główny z rysunku 5.2 a ma postać schematu sekwencyjnego, łatwego do analizy i zrozumienia. Rozpoczyna się od wczytania wartości *n*. W kolejnym bloku wywoływany jest podprogram *Silnia*, któremu jest przekazywana wczytana liczba naturalna. Po dokonaniu obliczeń następuje powrót z podprogramu, a wynik jest wyświetlany na ekranie. Cała złożoność obliczeniowa algorytmu przeniesiona jest do podprogramu przedstawionego na rysunku 5.2 b.

Oto, jak działa algorytm z rysunku 5.2 b dla $n = 3$:

- Wraz z wywołaniem funkcji *Silnia* jest do niej przekazywany argument $n = 3$. Ponieważ 3 jest różne od 0, wynikiem komparacji w bloku warunkowym jest odpowiedź negatywna. Zgodnie z formułą podaną w klatce wykonawczej funkcja przyjmuje, że jej wynikiem jest *3*Silnia(2)*. Jednak *Silnia(2)* nie jest znana, więc następuje chwilowe wstrzymanie obliczania wyrażenia *3*Silnia(2)* oraz uruchomienie (wywołanie) algorytmu dla $n = 2$.
- Algorytm wywołał sam siebie z argumentem $n = 2$. Obliczana jest wartość *Silnia(2)*. Ponieważ $2 > 0$, odpowiedzią w bloku warunkowym jest ponownie NIE. Podprogram uruchomi *Silnia(1)* i pomnoży ją przez dwa. Wartość wyniku cząstkowego *Silnia(1)* jest nieznana, dlatego następuje wstrzymanie obliczania wartości *2*Silnia(1)* i ponowne odwołanie do tej samej procedury rekurencyjnej z argumentem $n = 1$.
- Dla przekazanego argumentu $n = 1$ nadal nie jest spełniony warunek $n = 0$ i odpowiedzią komparatora jest NIE. *Silnia(1)* odwoła się zatem do kolejnej instancji podprogramu rekurencyjnego — uruchomi *Silnia(0)* i pomnoży ją przez jeden. Ponieważ wartość wyrażenia *Silnia(0)* w tym odwołaniu nie jest znana, obliczanie *1*Silnia(0)* zostaje wstrzymane, a podprogram rekurencyjny wykonuje swą kolejną bliźniaczą kopię z argumentem równym zero.
- Uruchomiony po raz kolejny podprogram wykonywany jest dla $n = 0$ i obliczana jest *Silnia(0)*. Wynikiem porównania argumentu z zerem jest odpowiedź twierdząca. Wykonywany jest blok, w którym *Silnia(0)* przyjmuje wartość 1.

- Skoro znany jest wynik *Silnia(0)*, może już nastąpić powrót z wywołań i obliczenie rzeczywistych wartości iloczynów. Znana już wartość *Silnia(0)* = 1 zostaje przekazana do instancji ją wywołującej i wówczas *Silnia(1)* = 1 · 1 = 1, analogicznie *Silnia(2)* = 2 · 1 i przyjmuje wartość dwa. Cofając się ponownie, otrzymujemy *Silnia(3)* = 3 · 2, co daje wynik końcowy równy 6, a to właśnie 3! = 1 · 2 · 3.

Zapamiętaj!
Wywoływanie kolejnych, bliźniaczych egzemplarzy podprogramu trwa dopóty, dopóki dla pewnego argumentu istnieje konkretny wynik cząstkowy.

W naszym algorytmie jest to wartość argumentu $n = 0$.

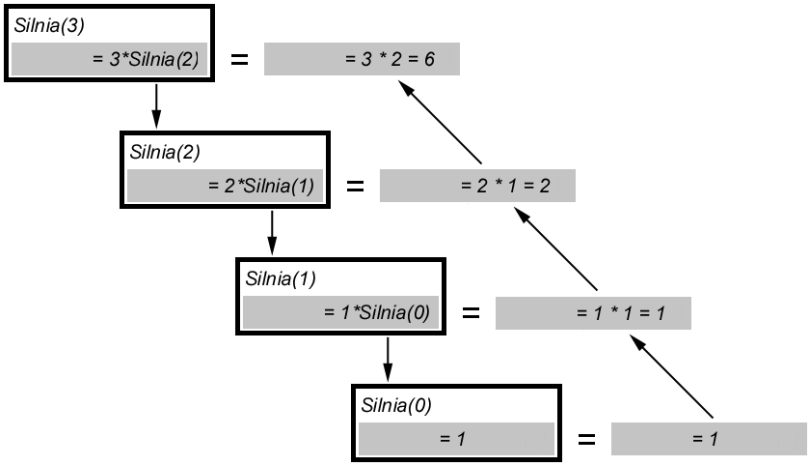
Poziomy i zagłębianie się

Każde kolejne wywołanie rekurencyjne odbywa się dla argumentu o 1 mniejszego niż w poprzednim egzemplarzu procedury rekurencyjnej. Każda wywołana instancja podprogramu rekurencyjnego nazywana jest **poziomem**. Kolejne poziomy identyfikowane są poprzez numer równy wartości *n*. **Poziom 0** oznacza elementarny egzemplarz procedury rekurencyjnej, podczas wykonania której uzyskuje się jednoznaczny wynik. Dopiero w chwili powrotu z wywołań obliczane są wyniki rzeczywiste. Z poziomu 0 wynik cząstkowy przekazywany jest na kolejne wyższe poziomy: poziom 1, poziom 2 itd.

Wywoływanie kolejnych rekurencyjnych egzemplarzy podprogramu nazywane jest **zagłębianiem** się z poziomu *n* na poziom $n - 1$. Przekazywanie informacji (danych wsadowych i wyników cząstkowych) odbywa się za pomocą pamięci komputerowej zwanej **stosem**. Więcej na ten temat znajduje się w uwagach końcowych do tego rozdziału.

Działanie opisanego powyżej algorytmu rekurencyjnego obliczającego *Silnia(3)* przedstawia rysunek 5.4.

Na rysunku 5.4 strzałka pionowa oznacza zagłębianie się algorytmu z poziomu wyższego na poziom niższy. Strzałka ukośna oznacza przekazanie wyniku cząstkowego z poziomu niższego na wyższy.



Rysunek 5.4. *Drzewo wywołań rekurencyjnych i przekazywania wyniku cząstkowego przy obliczaniu Silnia(3)*

ĆWICZENIE

5.2 Algorytm rekurencyjnego obliczania n!. Program w Pascalu

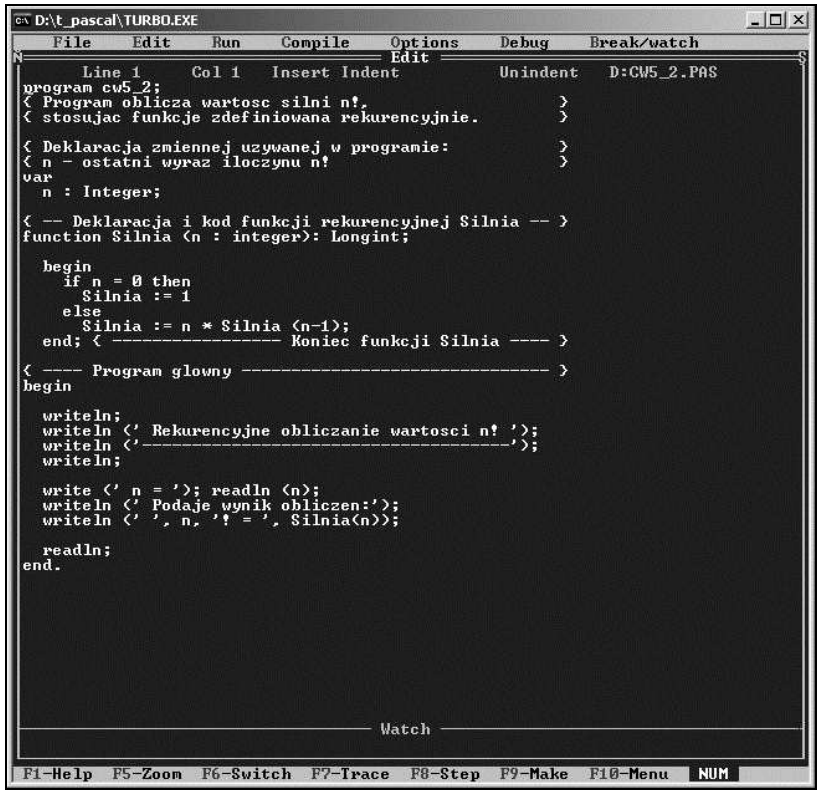
Wykorzystując algorytm z ćwiczenia 5.1, napisz rekurencyjny program w Turbo Pascalu, który obliczy i wyświetli wartość funkcji $n!$, dla $n \in N$.

Rozwiązanie

1. Uruchom Turbo Pascala i utwórz nowy plik, wybierając z paska menu polecenia *File*/*New*.
2. W oknie edycyjnym wpisz kod z listingu 5.1 albo wczytaj program z pliku *cw5_2.pas* znajdującego się w katalogu *TP*/*Rozdz_05*. Rezultat powinien być identyczny jak na rysunku 5.5.

Listing 5.1. *Kod rekurencyjnego programu obliczającego wartość silni*

```
program cw5_2;
{ Program oblicza wartosc silni n!,                    }
{ stosujac funkcje zdefiniowana rekurencyjnie.        }
```



Rysunek 5.5. *Okno edycyjne TP z kodem rekurencyjnego programu obliczania n!*

```
{ Deklaracja zmiennej uzywanej w programie:          }
{ n - ostatni wyraz iloczynu n!                      }
var
  n : Integer;

{ -- Deklaracja i kod funkcji rekurencyjnej Silnia -- }
function Silnia (n : Integer): Longint;

  begin
    if n = 0 then
      Silnia := 1
    else
      Silnia := n * Silnia (n-1);
  end; { ----------------- Koniec funkcji Silnia ---- }
```

```
{ ---- Program glowny ------------------------------ }
begin

  writeln;
  writeln (' Rekurencyjne obliczanie wartosci n! ');
  writeln ('-----------------------------------');
  writeln;

  write (' n = '); readln (n);
  writeln (' Podaje wynik obliczen:');
  writeln (' ', n, '! = ', Silnia(n));

  readln;
end.
```

Symbole i nazwy użyte w programie są identyczne jak w algorytmie z rysunku 5.2, dzięki czemu jego zrozumienie nie powinno sprawić kłopotu. W razie wątpliwości proszę jeszcze raz przeanalizować przykład poprzedni.

Najistotniejszym fragmentem programu jest rekurencyjna funkcja użytkownika o nazwie *Silnia*. Blok instrukcji ją tworzących funkcję rozpoczyna się deklaracją w postaci: `function Silnia (n : Integer): Longint`. Argument funkcji *n* jest liczbą całkowitą wprowadzaną przez użytkownika, a jej wynik jest typu *Longint*.

Funkcja wywoływana jest w głównym torze programu. Służy do tego komenda `Silnia(n)`, umieszczona w linii organizującej sposób wyświetlenia wyniku w postaci `writeln (n, '! = ', Silnia(n))`.

Wywołana funkcja działa zgodnie z przepływem na schemacie z rysunku 5.2 b. Obliczenia rekurencyjne zostały zrealizowane za pomocą bloku warunkowego. Jeżeli $n > 0$, to wykonywana jest instrukcja rekursyjna `Silnia := n * Silnia (n-1)`. Kolejne odwołania trwają tak długo, aż argument funkcji zyska wartość równą zero. Oznacza to, że został osiągnięty poziom zerowy zagłębienia w podprogram. Uzyskany na tym poziomie wynik cząstkowy jest konkretną liczbą i może być przekazany na poziom wyższy, gdzie następują kolejne obliczenia. Na najwyższym poziomie *n* obliczana jest wartość stanowiąca wynik końcowy wyświetlany na ekranie (rysunek 5.6).

Rysunek 5.6. *Efekt wykonania programu cw5_2*

ĆWICZENIE

5.3 Aplikacja rekurencyjnego obliczania silni w Excelu

Napisz w Excelu aplikację obliczającą rekurencyjnie silnię *n*!. W tym celu utwórz funkcję użytkownika działającą według algorytmu z rysunku 5.2 b.

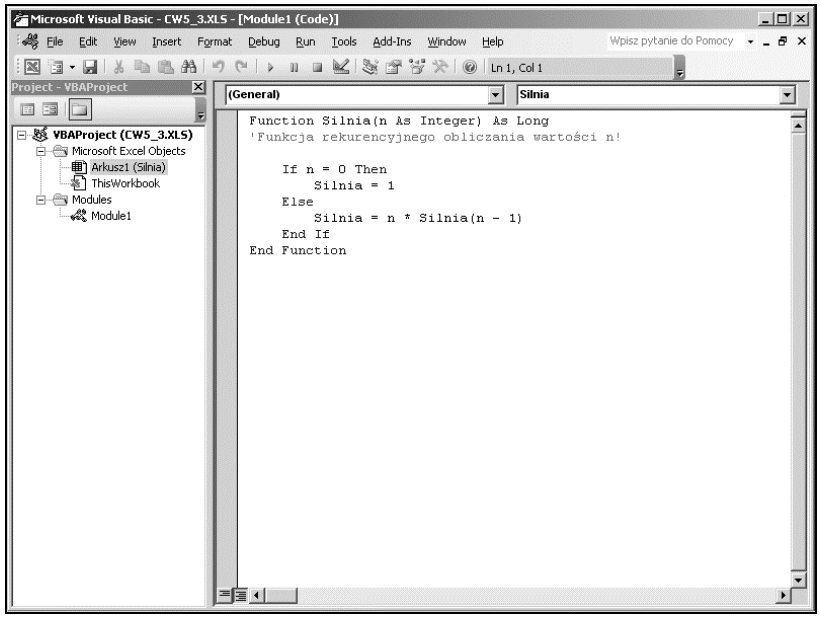
Rozwiązanie

1. Uruchom program Excel i zapisz domyślnie pojawiający się *Zeszyt1* w wybranym przez siebie katalogu pod nazwą *cw5_3*. Można również wczytać arkusz *cw5_3.xls* z katalogu *EX/Rozdz_05*.
2. Zmień nazwę zakładki *Arkusz1* na *Silnia*.
3. Usuń zakładki *Arkusz 2* i *Arkusz3*.
4. W komórce *C2* umieść tekst: *Aplikacja rekurencyjnego obliczania silni n!*. Proponowana czcionka: Arial CE, pogrubiona, w kolorze niebieskim, rozmiar 18.
5. Wprowadź funkcję przeliczeniową *Silnia*. W tym celu:
   - Wywołaj okno edytora VBE i wstaw moduł standardowy *Module1* (Moduł1).
   - W sekcji *General* (Ogólne) modułu *Module1* (Moduł1) wpisz kod z listingu 5.2. Powinieneś uzyskać efekt jak na rysunku 5.7.

Listing 5.2. *Funkcja użytkownika Silnia w ćwiczeniu cw5_3*

```
Function Silnia(n As Integer) As Long
'Funkcja rekurencyjnego obliczania wartości n!

    If n = 0 Then
        Silnia = 1
    Else
        Silnia = n * Silnia(n - 1)
    End If
End Function
```



Rysunek 5.7. *Wygląd okna edytora VBE z wpisaną funkcją Silnia*

Wprowadzona funkcja jest bliźniaczo podobna do funkcji utworzonej w ćwiczeniu poprzednim. Działa również identycznie. Jedynie znaczniki początku i końca nieco się od siebie różnią.

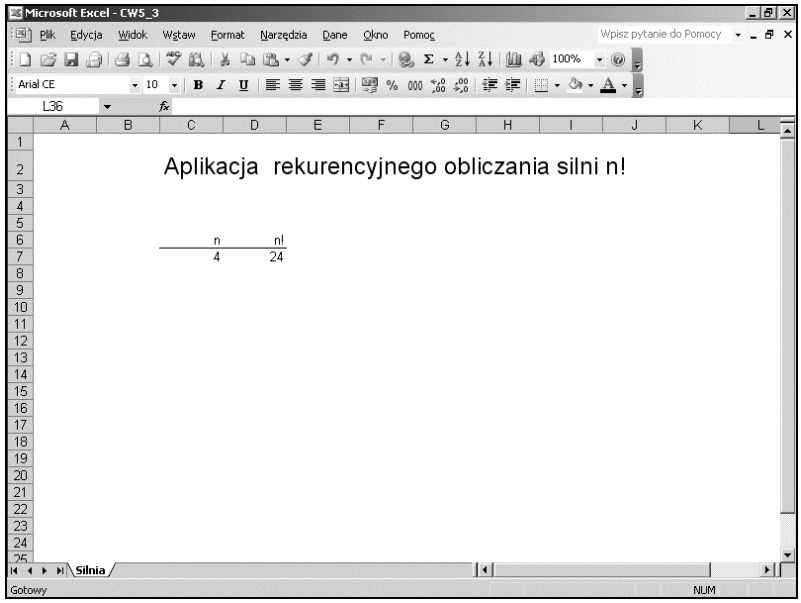
6. Dokończ budowę tabeli arkusza, wykonując podane poniżej polecenia:
   - We wskazanych komórkach arkusza umieść nagłówki:

- komórka *C6* — *n*,
- komórka *D6* — *n*!,
- komórka *C7* — wpisz liczbę 4.

Proponowana czcionka: Arial CE, normalna, rozmiar 10. Wyrównaj do prawej zawartość *C6:D6* oraz podkreśl komórki stylem *Krawędź dolna*.

- Wpisz w komórce *D7* formułę wywołującą funkcję: *=SILNIA(C7)*. Możesz również skorzystać z menu *Wstaw*, kliknąć polecenie *Funkcja…* i wybrać funkcję użytkownika o nazwie *Silnia*. Jako jej argument należy podać komórkę *C7*.
- Wyłącz siatkę arkusza.

Zakończyłeś tworzenie arkusza, który powinien mieć wygląd jak na rysunku 5.8.



Rysunek 5.8. *Arkusz aplikacji cw5_3*

Sprawdź działanie aplikacji. Poeksperymentuj, zmieniając wartości w komórce *C7*, a następnie zakończ pracę z arkuszem i Excelem, wybierając *Plik* oraz *Zakończ*.

Obliczanie potęgi liczby rzeczywistej

Zagadnienie obliczania potęg zostało już zasygnalizowane w ćwiczeniu 2.1 podczas omawiania algorytmów sekwencyjnych. Rozważania dotyczyły jednak tylko potęg z wykładnikiem parzystym. Obecnie zostanie przedstawiona rekurencyjna metoda obliczania wartości potęgi o dowolnym wykładniku. Przykład zobrazuje jednocześnie, jak w jednym podprogramie użyć dwóch instrukcji rekurencyjnych.

ĆWICZENIE

5.4 Rekurencyjne obliczanie potęgi liczby rzeczywistej

Przedstaw w postaci listy kroków rekurencyjny algorytm funkcji obliczającej potęgę a^n, gdzie $a \in R$, $n \in N$.

Rozwiązanie

Dane: Wartość podstawy $a \in R$ oraz potęgi $n \in N$.

Oczekiwany wynik: Wartość podstawy (argumentu) a podniesionej do potęgi n.

Analiza problemu: Potęgowanie rekurencyjne bazuje na podnoszeniu liczby do kwadratu.

Dla $n = 1$ wynikiem obliczeń jest wartość podstawy a.

Dla $n > 1$ pierwsze działanie zależy od tego, czy wykładnik jest parzysty, czy nie:

- Jeżeli wykładnik jest liczbą naturalną parzystą, to doprowadza się go do takiej postaci, by występowało potęgowanie wewnętrzne i zewnętrzne o wykładniku 2, na przykład $3^4 = (3^2)^2$, $2^{10} = (2^5)^2$. Dla dowolnej parzystej liczby n, zapis ten ma postać:

$$a^n = (a^{\frac{n}{2}})^2 \cdot$$

- Jeżeli wykładnik jest nieparzysty większy od jedności, to wyodrębnia się fragment z potęgą parzystą i otrzymany wynik pośredni mnoży się przez podstawę a, na przykład $3^9 = 3^8 \cdot 3$. Dla dowolnej liczby nieparzystej n, zapis ten ma postać:

$$a^n = a^{n-1} a \cdot$$

Teraz wykładnik $n - 1$ we wzorze jest już parzysty, zatem potęgowanie można zapisać w postaci:

$$a^n = (a^{\frac{n-1}{2}})^2 a \cdot$$

Operacje redukowania należy powtarzać tak długo, aż wszystkie działania w wyrażeniu otrzymają opisaną wyżej postać. Obrazują to przykłady: $3^9 = 3^8 \cdot 3 = (3^4)^2 \cdot 3 = ((3^2)^2)^2 \cdot 3$, $7^{14} = (7^7)^2 = (7^6 \cdot 7)^2 = ((7^3)^2 \cdot 7)^2 = ((7^2 \cdot 7)^2 \cdot 7)^2$.

Skoro za każdym razem istotna jest informacja, czy podstawa jest parzysta, czy nieparzysta, to w algorytmie musi wystąpić fragment, który sprawdza parzystość wykładnika. W tym celu wystarczy podzielić liczbę będącą wykładnikiem przez 2. Jeżeli reszta z dzielenia równa jest zero, to wykładnik jest podzielny przez 2, a reszta ma wartość zero.

Drugim stałym elementem w zredukowanych wyrażeniach jest podnoszenie do kwadratu. Warto tę operację zrealizować za pomocą odrębnej funkcji, do której przekazuje się odpowiedni argument.

Po uwzględnieniu parzystości i dokonaniu redukcji wykładnika według reguł podanych powyżej otrzymujemy zależność klamrową w postaci:

$$a^n = \begin{cases} a, & \text{dla } n = 1 \quad (5.1) \\ (a^{\frac{n}{2}})^2, & n \text{ jest liczbą parzystą} \quad (5.2) \\ (a^{\frac{n-1}{2}})^2 a, & n \text{ jest liczbą nieparzystą} \quad (5.2) \end{cases}$$

Algorytm w postaci listy kroków

Zakładamy, że tworzymy dwuargumentową funkcję o nazwie *Potega*, do której przekazywane są następujące argumenty: *podstawa* — dowolna liczba rzeczywista $a \in R$, *wykładnik* — liczba naturalna $n \in N$. Postać funkcji rekurencyjnej jest zatem dwuargumentowa: *Potega*(a, n). Funkcja ta wywoływana jest każdorazowo, gdy wystąpi w algorytmie.

Krok 1. Sprawdź, czy $n = 1$. Jeżeli tak, to podstaw *Potega* = a, po czym przejdź do kroku 7. Jeżeli nie, to przejdź do kroku 2.

Krok 2. Sprawdź, czy reszta z dzielenia wykładnika n przez 2 jest równa zero. Jeżeli tak, to przejdź do kroku 3. Jeżeli nie, to przejdź do kroku 5.

Krok 3. {Wykładnik jest liczbą parzystą.} Przypisz $n = n/2$ i przejdź do kroku 4.

Krok 4. {Obliczanie potęgi liczby a zgodnie ze wzorem (5.2) z zależności klamrowej podanej powyżej}. Wywołaj funkcję rekurencyjną *Potega*(a, n), a następnie podnieś ją do kwadratu: *Potega* = (*Potega* (a, n))^2. Przejdź do kroku 7.

Krok 5. {Wykładnik jest liczbą nieparzystą.} Podstaw $n = (n - 1)/2$ i przejdź do kroku 6.

Krok 6. {Obliczanie potęgi liczby a zgodnie ze wzorem (5.3) z zależności klamrowej.} Wywołaj funkcję *Potega*(a, n), po czym podnieś ją do potęgi drugiej i pomnóż przez podstawę a: *Potega* = (*Potega*(a, n))^2*a. Przejdź do kroku 7.

Krok 7. Zakończ działanie algorytmu. Wynikiem jest bieżąca wartość *Potega*.

Sprawdź — wykonując obliczenia na papierze — poprawność algorytmu dla wybranych wartości a oraz n.

ĆWICZENIE

5.5 Algorytm rekurencyjnego obliczania potęgi. Program w Turbo Pascalu

Napisz w Turbo Pascalu program rekurencyjnego obliczania potęgi naturalnej dowolnej liczby rzeczywistej. W programie wykorzystaj funkcję zbudowaną z wykorzystaniem algorytmu przedstawionego w ćwiczeniu 5.4. Podnoszenie do kwadratu wykonaj za pomocą funkcji elementarnej *Sqr*.

Rozwiązanie

Funkcja zrealizowana według opisu podanego w algorytmie z ćwiczenia 5.4 nie zawiera bloku wprowadzania danych i wyświetlania wyniku. Odpowiednie, umożliwiające to instrukcje muszą znaleźć się w programie głównym, z którego nastąpi wywołanie funkcji potęgującej.

1. Uruchom Turbo Pascala i utwórz nowy plik, wybierając z paska menu polecenia *File*/*New*.
2. W oknie edycyjnym wpisz kod z listingu 5.3 albo wczytaj program z pliku *cw5_5.pas* znajdującego się w katalogu *TP*/*Rozdz_05*.

Listing 5.3. *Kod rekurencyjnego programu obliczającego wartość naturalnej potęgi liczby rzeczywistej*

```
program cw5_5;
{ Program oblicza rekurencyjnie wartosc               }
{ liczby a podniesionej do potegi n.                  }

{ Deklaracja zmiennych uzywanych w programie:         }
{ a - liczba potegowana, n - wykladnik potegi.        }
var
   a: Real; n: Integer;

{ ---- Deklaracja i kod funkcji rekurencyjnej Potega ------ }
function Potega (a: Real; n : Integer): Real;

  begin
    if n = 1 then
      Potega := a
    else
      if (n mod 2 = 0) then
        begin
          n := n div 2;
          Potega :=  Sqr( Potega(a, n));
        end
      else
        begin
          n := (n - 1) div 2;
          Potega := Sqr(Potega(a, n)) * a;
        end
  end; { ----------------- Koniec funkcji Potega ---- }

{ ---- Program glowny ------------------------------ }
begin

  writeln;
  writeln (' Rekurencyjne obliczanie potegi podanej liczby ');
  writeln ('-------------------------------------------------');
  writeln;
```

```
  write (' Podstawa a = '); readln (a);
  write (' Wykladnik n = '); readln (n);
  writeln;
  writeln (' Wynik obliczen: ');
  writeln (' ', a:0:2, ' do potegi ', n, ' = ', Potega(a,n):0:2);

  readln;
end.
```

Funkcja rekurencyjna *Potega* występująca w listingu 5.3 jest dokładnym odwzorowaniem algorytmu i tak też działa. Do podnoszenia do kwadratu służy funkcja wbudowana `Sqr(argument)`, która oblicza kwadrat podanego w nawiasie argumentu.

Sprawdzenie parzystości liczby dokonywane jest w instrukcji warunkowej przy wykorzystaniu instrukcji `mod` o składni: `n mod 2`. Wynikiem tej operacji jest reszta z dzielenia liczby całkowitej *n* przez 2. Rezultat zero oznacza, że *n* jest podzielne przez 2 — jest zatem liczbą parzystą i wykonywany jest blok instrukcji po słowie kluczowym `then`. W przypadku *n* nieparzystego program wykonuje polecenia po słowie *else*.

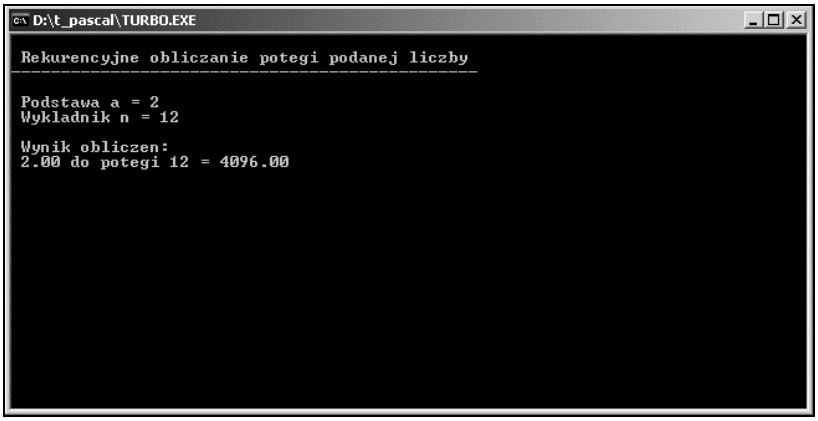
Iloraz w podprogramie obliczany jest za pomocą funkcji `div`, która realizuje dzielenie całkowite liczb całkowitych. Oznacza to, że nie występuje reszta z dzielenia, na przykład `7 div 4 = 1`. Wynik dzielenia jest przypisywany argumentowi *n*, który jest liczbą naturalną.

Główny tor programu to deklaracja zmiennych oraz wczytanie danych: podstawy *a* i wykładnika *n*. Potem wywoływana jest dwuargumentowa funkcja *Potega*(*a*, *n*). Wywołanie następuje bezpośrednio z linii wyprowadzającej wyniki na ekran: `writeln (a:0:2, ' do potegi ', n, ' = ', Potega(a,n):0:2)`. Sposób wyświetlania danych i rezultatu obliczeń — z dwoma miejscami dziesiętnymi — można oczywiście dostosować według uznania. Efekt wykonania programu przedstawia rysunek 5.9.

ĆWICZENIE

5.6 Algorytm rekurencyjnego obliczania potęgi. Aplikacja w Excelu

Napisz w Excelu program rekurencyjnego obliczania potęgi naturalnej dowolnej liczby rzeczywistej. W programie wykorzystaj funkcję użytkownika zbudowaną z wykorzystaniem algorytmu przedstawionego w ćwiczeniu 5.4.



Rysunek 5.9. *Efekt wykonania programu cw5_5*

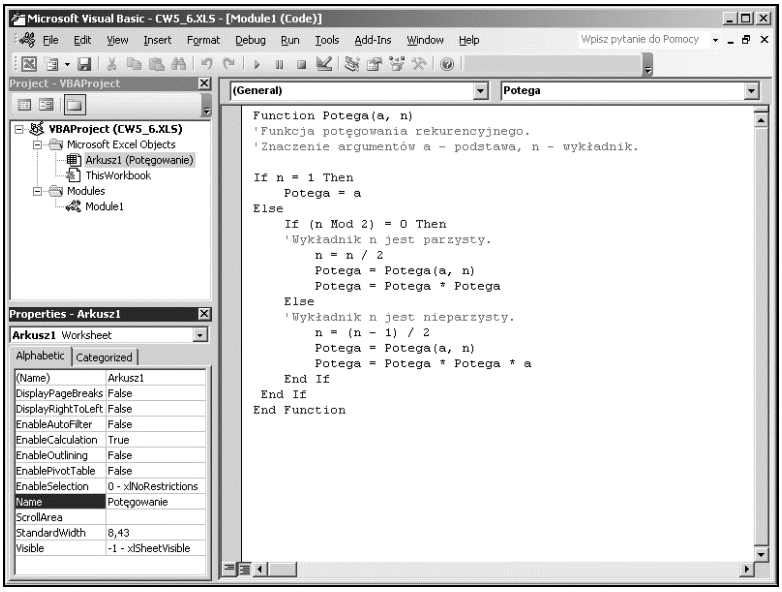
Rozwiązanie

1. Uruchom program Excel i zapisz domyślnie pojawiający się *Zeszyt1* w wybranym przez siebie katalogu pod nazwą *cw5_6* albo wczytaj arkusz *cw5_6.xls* z katalogu *EX/Rozdz_05*.
2. Zmień nazwę zakładki *Arkusz1* na *Potęgowanie*.
3. Usuń zakładki *Arkusz 2* i *Arkusz3*.
4. W komórce *C2* umieść tekst — *Aplikacja rekurencyjnego obliczania potęgi*. Proponowana czcionka: Arial CE, pogrubiona, w kolorze fioletowym, rozmiar 18.
5. Utwórz funkcję przeliczeniową *Potega*. W tym celu:
   - Wywołaj okno edytora VBE i wstaw moduł standardowy *Module1* (*Moduł1*).
   - W sekcji *General* (*Ogólne*) modułu *Module1* (*Moduł1*) wpisz kod z listingu 5.4, tak jak przedstawia to rysunek 5.10.

Listing 5.4. *Kod funkcji Potega z ćwiczenia 5.6*

```
Function Potega(a, n)
'Funkcja potęgowania rekurencyjnego.
'Znaczenie argumentów a - podstawa, n - wykładnik.

If n = 1 Then
    Potega = a
Else
    If (n Mod 2) = 0 Then
```



Rysunek 5.10. *Edytor VBE z kodem funkcji Potega. Po lewej widoczne jest okno eksploratora, a poniżej okno właściwości budowanego arkusza Potęgowanie*

```
'Wykładnik n jest parzysty.
        n = n / 2
        Potega = Potega(a, n)
        Potega = Potega * Potega
    Else
    'Wykładnik n jest nieparzysty.
        n = (n - 1) / 2
        Potega = Potega(a, n)
        Potega = Potega * Potega * a
    End If
 End If
End Function
```

Działanie funkcji jest identyczne jak w ćwiczeniu poprzednim. Niewielkie różnice w kodzie polegają na innym zorganizowaniu podnoszenia do kwadratu (mnożenie przez siebie) oraz na zastosowaniu zwykłego operatora dzielenia (/).

6. Dokończ budowę arkusza, tworząc tabelę przeliczeniową:

- We wskazanych komórkach arkusza umieść nagłówki:

- komórka *C4* — `Podstawa a`,
- komórka *E4* — `Wykładnik n`,
- komórka *G4* — `an`; sformatuj literę *n* jako *Indeks górny*,
- komórka *C5* — `2`,
- komórki *E5:E14* — wprowadź kolejne liczby naturalne od `1` do `10`.

- Zmień szerokość kolumn *C*, *E*, *G* na 85 pikseli.
- Podkreśl komórki arkusza *C4*, *E4* i *G4* stylem *Gruba krawędź dolna*,. Zmień kolor tekstu w komórkach na zielony, po czym go wyśrodkuj.

7. W komórce *G5* wpisz formułę przeliczeniową — *=Potega (C5;E5)*, a następnie skopiuj ją do komórek *G6:G14*.

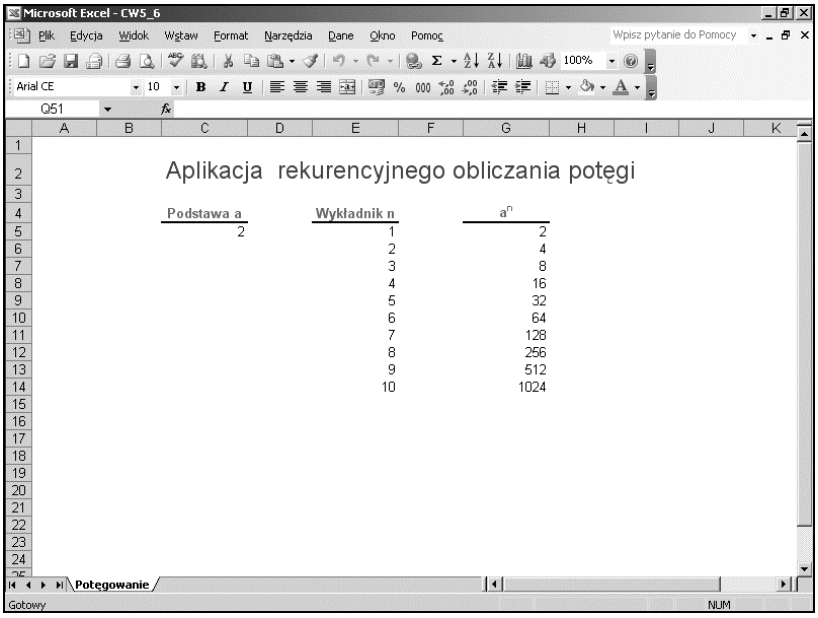
   Znak (*$*) oznacza adresowanie bezwzględne (absolutne) — podczas kopiowania formuły adres komórki *C5*, do której odwołuje się formuła, nie ulegnie zmianie. W formule występuje też odwołanie względne, które we wklejanej formule jest aktualizowane i dotyczy innych komórek względem położenia formuły. W naszej funkcji są to kolejne komórki z kolumny *E*, poczynając od *E5*.
8. Tworzenie arkusza zostało zakończone. Efekt widoczny jest na rysunku 5.11.
9. Poeksperymentuj z wartościami podstawy *a* oraz wykładnika *n*, zmieniając wartości w odpowiednich komórkach, a następnie zakończ pracę z arkuszem i Excelem, wybierając *Plik* oraz *Zakończ*.

Uwagi końcowe

Mocne i słabe strony rekurencji

Zalety programów realizowanych rekurencyjnie:

- pozwalają rozwiązywać problemy o dowolnej rozpiętości zbioru i trudne do opisu,
- zwięzłość i elegancja zapisu.



Rysunek 5.11. *Arkusz Potęgowanie z aplikacji cw5_6*

Niestety, są też poważne wady. Zaliczamy do nich:

- powielanie tych samych obliczeń,
- niejasny i trudny do analizy przebieg wywołań,
- niebezpieczeństwo nieskończonej liczby odwołań,
- duże zapotrzebowanie na pamięć podczas przetwarzania.

Niedogodności są spowodowane głównie tym, że po każdym odwołaniu rekurencyjnym zachodzi konieczność zapamiętania informacji potrzebnych do odtworzenia stanu procesu sprzed wywołania. Zapamiętywane informacje przechowywane są w obszarze pamięci zwanym stosem.

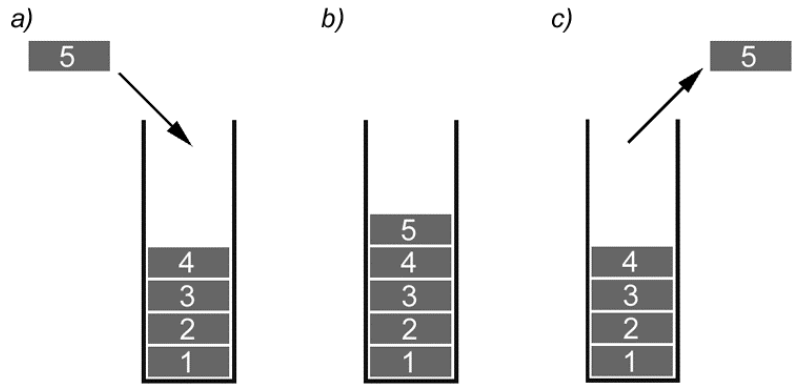
Stos

Stos (ang. *stack*) to obszar wewnętrznej pamięci komputerowej przeznaczonej do czasowego przechowywania informacji związanych z wykonywanym programem. Dla rekurencji istotne jest, by stos

posiadał strukturę **LIFO** (akronim z ang. ***L**ast **I**n **F**irst **O**ut*). W dosłownym tłumaczeniu oznacza **ostatni na wejściu jest pierwszym na wyjściu**. Komputer odzyskuje potrzebne do wykonania programu informacje, pobierając je z wierzchołka stosu. Żądany element lokalizowany jest dzięki rejestrowi zwanemu **wskaźnikiem stosu** (ang. *stack pointer*), który jest powiększany o 1 każdorazowo przed umieszczeniem kolejnego elementu na stosie i dekrementowany o 1 po zdjęciu elementu ze stosu. Łatwo zauważyć, że gdy wskaźnik ma wartość zero, to stos jest pusty.

Stos jest obszarem pamięci o ograniczonej pojemności, dlatego łatwo może dojść do jego przepełnienia. Podczas rekursji zdarza się to nader często i wywołuje błąd, który sygnalizowany jest komunikatem **stack overflow** (z ang. *przepełnienie stosu*).

Działanie stosu obrazuje rysunek 5.12.



Rysunek 5.12. *Poglądowa struktura stosu obrazująca: a) dodanie informacji, b) przechowanie informacji, c) pobranie informacji ze stosu*

Z rysunku widać, że stos ma strukturę studni. Dane umieszczane są zawsze na szczycie stosu i stąd też pobierane. Informacja wprowadzona jako pierwsza zostanie pobrana jako ostatnia.

Ćwiczenia do samodzielnego wykonania

ĆWICZENIE **5.7**

Ułóż algorytm obliczania sumy kolejnych liczb naturalnych.

ĆWICZENIE **5.8**

Sprawdź, czy w podprogramie z listingu 5.3 można zastosować zwykły operator dzielenia (/). Czy instrukcję `n := (n - 1) div 2` można zastąpić poleceniem `n := n div 2`? Jak to wyjaśnić?

ĆWICZENIE **5.9**

Przedstaw algorytm z ćwiczenia 5.4 w postaci schematu blokowego.

ĆWICZENIE **5.10**

Ułóż algorytm obliczania pierwiastka stopnia *n* z podanej liczby, a następnie zakoduj go w Turbo Pascalu i Visual Basicu.