Rozdział 12. Rekurencja

Rekurencja (rekursja) jest to metoda rozwiązywania problemu za pomocą odwoływania się do definicji. W programowaniu jest ona możliwa dzięki funkcjom. Idea działania rekurencji polega na tym, że mając złożone zadanie (złożoność może być przykładowo opisywania dużą wartością wybranego parametru) wykonujemy pewne, znane, operacje i ponownie rozwiązujemy zadanie ale o mniejszym stopniu złożoności. Przykładowo, chcemy policzyć 5! – przypuśćmy, że jest to zadanie złożone. Ale mamy:

5!=5\*4!.

Powyższy wzór pokazuje, że chcąc obliczyć złożone wyrażenie – tu 5! - wykonaliśmy pewną operację - tu mnożenie - i dalej musimy wykonać operację mniej złożoną – tu 4!. Postępując analogicznie w podobny sposób mamy:

4!=4\*3!=4\*(3\*2!)=4\*(3\*(2\*1!))=4\*(3\*(2\*(1\*0!))).

Dalsze „upraszczanie nie ma już sensu, bowiem wartość „0!” znamy z definicji – więc nic nie musimy obliczać.

 Powyższe postępowanie pokazuje nam jeszcze jedną cechą rekurencji, proces „upraszczania” problemu musimy kiedyś przerwać. Jest to nazywane warunkiem stopu. Podsumowując, możemy powiedzieć, że rekurencja w programowaniu polega na wywoływaniu funkcji przez samą siebie do sytuacji, w której możemy rozwiązać dany problem na podstawie definicji (definicja oznacza, że poszukiwana wielkość jest znana, nie ma potrzeby jej obliczania).

12.1. Schemat działanie rekurencji

Rozważmy poniższy program, listing 12.1.

#include <iostream>

using namespace std;

void p(int i)

{

 if (i>0)

 {

 cout<<"i="<<i<<endl;

 p(i-1);

 return;

 }

 return;

}

int main()

{

 p(4);

 system("pause");

 return 1;

}

Listing 12.1. Przykład rekurencji

Wynik działania tego programu został pokazany na rys. 12.1.



Rys.12.1. Wynik działania programu pokazanego na listingu 12.1.

Aby ułatwić zrozumienie działania tego programu zostaną wypisane instrukcje lub ich działanie kolejno wywoływane przez niego:

main()

 p(4)

 void p(4)

 if (4>0) //tak

 i=4

 p(4-1)

 void p(3)

 if (3>0) //tak

 i=3

 p(3-1)

 void p(2)

 if (2>0) //tak

 i=2

 p(2-1)

 void p(1)

 if (1>0) //tak

 i=1

 p(1-1)

 void p(0)

 if (0>0) //nie

 return //p(0)

 return //p(1)

 return //p(2)

 return //p(3)

 return //p(4)

return 1 //main()

Jak widać z powyższego schematu funkcja p jest kolejno wywoływana wtedy, gdy warunek “i>0” jest prawdziwy. Jeśli warunek jest nieprawdziwy, to następuje powrót do funkcji wołającej, która również wraca do swojej wołającej (w obu przypadkach są to funkcje p) itd.

12.2. Porównanie rekurencji z rozwiązaniami iteracyjnymi

 Łatwo pokazać, że dysponując rozwiązaniem iteracyjnym możemy stosunkowo łatwo uzyskać rekurencyjne. W tym celu rozważmy problem:

Oblicz sumę liczb całkowitych zakończonych zerem.

Dla pętli do while fragment tekstu programu może wyglądać następująco:

 int s=0,x;

 do

 {

 cout<<"x=";cin>>x;

 s=s+x;

 } while (x!=0);

 cout<<"suma="<<s;

Natomiast możliwe rozwiązanie rekurencyjne dla pętli do while zostało pokazane na listingu 12.2.

#include <iostream>

using namespace std;

int pdo\_while(int warunek)

{

 int x;

 if (warunek)

 {

 cout<<"x=";cin>>x;

 return x+pwhile(x!=0);

 }

 else return 0;

}

int main()

{

 cout<<"wynik:"<<pdo\_while(1!=0)<<endl;

 system ("pause");

}

Listing 12.2. Rozwiązanie rekurencyjne problemu „sumowanie liczb do napotkania zera”

Wynik działania tego programu został pokazany na listingu 12.2.



Listing 12.2.Wynik działania programu, którego tekst znajduje się na listingu 12.2

Jak widać, wynik działania programu jest zgodny z oczekiwanym wynikiem. Podobnie można podać rozwiązania rekurencyjne dla innych pętli.

 Taka sytuacja oznacza, że mając rozwiązanie iteracyjne mamy także rozwiązanie rekurencyjne. W takiej sytuacji naturalnym jest pytanie, czy mając rozwiązanie rekurencyjne mamy także rozwiązanie iteracyjne. Okazuje się, że nie zawsze. Ogólnie można powiedzieć, że w sytuacji, gdy jeśli liczba pętli jest uzależniona od stopnia złożoności problemu, to rozwiązanie iteracyjne nie można być skonstruowane (problem N-skoczków, N-hetmanów). Jest to konsekwencją jednego z postulatów von Neumana, że tekst programu nie może zależeć od danych. Jeden z przykładów zostanie pokazany w dalszej części tego rozdziału.

 12.3. Rekurencyjne rozwiązanie dla wież Hanoi

 Problem wież Hanoi, rys. 12.3 polega na tym, że mamy trzy słupki, oznaczmy je jako A, B i C. Na początku na słupku A znajdują się n krążków, ułożonych zgodnie z zasadą, że na mniejszym słupku może leżeć większy słupek. Zadanie polega na tym, by ze słupka A przenieść wszystkie krążki na słupek C zachowując podaną wcześniej zasadę.



Rys. 12.3 Problem wież Hanoi, źródło: Internet

Rozwiązanie iteracyjne jest bardzo trudne, a rozwiązanie rekurencyjne jest bardzo proste. Ideę rozwiązania można przedstawić następująco:

* Problem zdefiniujmy jako: Hanoi(n,'A','C','B'), tzn. przenieś krążki z 'A' na 'C' z wykorzystaniem ,'B',
* Przyjmując, że jest to trudny problem przenosimy:
	+ n-1 górnych krążków z 'A' na 'B' z wykorzystaniem 'C', ten problem jest zdefiniowany jako: Hanoi(n,'A','B','C'),
	+ pojedynczy krążek z 'A' przenosimy na 'C' – jest to proste zadanie,
	+ n-1 krążków z 'B' na 'C' z wykorzystaniem 'A', ten problem jest zdefiniowany jako Hanoi(n,'B','C','A').

Tekst programu bazującego na powyższym schemacie został pokazany na listingu 14.4.

#include <iostream>

using namespace std;

void Hanoi(int n, char a, char c, char b)

{

 if (n>0)

 {

 Hanoi(n-1,a,b,c);

 cout<<a<<"->"<<c<<endl;

 Hanoi(n-1,b,c,a);

 }

}

int main()

{

 int n;

 cout<<"n=";cin>>n;

 Hanoi(n,'A','C','B');

 return 0;

}

Listing 14.4. Rozwiązanie rekurencyjne dla wież Hanoi

Wynik działania programu wykonanego na podstawie tekstu programu dla n=4 został pokazany na rys. 14.5



Rys. 12.5. Wynik działania programu dla tekstu programu pokazanego na listingu 12.4

Jak widać, wynik są poprawne. Należy mieć na uwadze przy uruchamianiu programu dla dużych n, że liczba wszystkich operacji wynosi 2n-1.

 12.3. Rekurencyjne rozwiązanie dla skoczków

Zadanie ze skoczkami polega na tym, że na szachownicy o wymiarze NxN należy, startując z pewnego pola na szachownicy, postawić N2 skoczków, tak by każdym polu był ustawiony dokładnie jeden skoczek.

 Rozwiązując to zadanie można poszukać rozwiązania bazując na następującej idei zapisanej w postaci pseudokodu (na początku (x,y) jest początkową pozycją skoczka:

* stawiamy skoczka pozycji S(x,y),
* zapisujemy numer skoczka na szachownicy i z pozycji S(x,y):
	+ sprawdzamy kolejno 8 możliwych pozycji (wynikają one ze schematu ruchu skoczka): (x+1, y-2), (x+2, y-1), (x+2, y+1), (x+1, y+2), (x-1, y+2), (x-2, y+1), (x-2, y-1), (x-1, y-2). Jeśli pole istnieje na szachownicy i jest puste to dla każdej pozycji:
		- zapisujemy numer skoczka na szachownicy i z pozycji S(x,y):
			* sprawdzamy kolejno 8 możliwych pozycji (wynikają one ze schematu ruchu skoczka): (x+1, y-2), (x+2, y-1), (x+2, y+1), (x+1, y+2), (x-1, y+2), (x-2, y+1), (x-2, y-1), (x-1, y-2). Jeśli pole istnieje na szachownicy i jest puste to dla każdej pozycji:

…

* w przeciwnym razie sprawdzamy, czy ustawione do tej pory skoczki spełniają warunki zadania, jeśli tak to drukujemy zawartość, jeśli nie to rozpatrujemy kolejną iterację.

Podany wyżej schemat działa w ten sposób, że zaczynając od pozycji startowej modyfikujemy szachownicę zaznaczając, że skoczek tu był i próbujemy iteracyjne dostawiać kolejne skoczki, a na końcu sprawdzamy czy są spełnione warunki zadania. Jeśli tak, drukujemy zawartość szachownicy, jeśli nie rozpatrujemy kolejny wariant (iterację).

Rozwiązanie rekurencyjne jest pokazane na listingu 11.5

#include <iostream>

#include <iomanip>

using namespace std;

int NR=1;

void init(int n,int \*\*s)

{

 for (int i=0;i<n;i++)

 for(int j=0;j<n;j++)

 s[i][j]=0;

}

void wypisz(int n, int \*\*s)

{

 cout<<"\n ruchy skoczka: rozwiazanie "<<NR;

 for (int i=0;i<n;i++)

 {

 cout<<endl;

 for (int j=0;j<n;j++)

 {

 cout.width(3);

 cout<<s[i][j]<<" ";

 }

 }

 cout<<endl;

 NR++;

// system("pause");

}

int spr(int poz, int wym)

{

 if (poz>=0 && poz <wym) return 1;

 else return 0;

}

int spr\_skoczki(int wym, int \*\*s)

{

 int sa=1;

 for(int i=0;i<wym;i++)

 for(int j=0;j<wym;j++)

 if (s[i][j]==0) sa=0;

 return sa;

}

int skoczek(int n, int ruch, int i, int j, int \*\*s)

{

 s[i][j]=ruch;

// wypisz(n,s);

 if (!spr\_skoczki(n,s))

 {

 if (spr(i+1,n)&&spr(j-2,n)&&s[i+1][j-2]==0)

 if (skoczek(n,ruch+1,i+1,j-2,s)==1) return 1;

 if (spr(i+2,n)&&spr(j-1,n)&&s[i+2][j-1]==0)

 if (skoczek(n,ruch+1,i+2,j-1,s)==1) return 1;

 if (spr(i+2,n)&&spr(j+1,n)&&s[i+2][j+1]==0)

 if (skoczek(n,ruch+1,i+2,j+1,s)==1) return 1;

 if (spr(i+1,n)&&spr(j+2,n)&&s[i+1][j+2]==0)

 if (skoczek(n,ruch+1,i+1,j+2,s)==1) return 1;

 if (spr(i-1,n)&&spr(j+2,n)&&s[i-1][j+2]==0)

 if (skoczek(n,ruch+1,i-1,j+2,s)==1) return 1;

 if (spr(i-2,n)&&spr(j+1,n)&&s[i-2][j+1]==0)

 if (skoczek(n,ruch+1,i-2,j+1,s)==1) return 1;

 if (spr(i-2,n)&&spr(j-1,n)&&s[i-2][j-1]==0)

 if (skoczek(n,ruch+1,i-2,j-1,s)==1) return 1;

 if (spr(i-1,n)&&spr(j-2,n)&&s[i-1][j-2]==0)

 if (skoczek(n,ruch+1,i-1,j-2,s)==1) return 1;

// s[i][j]=0;cout<<"cofamy: "<<i<<" "<<j<<endl;

 } else

 wypisz(n,s);

 s[i][j]=0;

 return 0;

}

int main()

{

 int n;

 cout<<"n=";cin>>n;

 int \*\*s=new int\*[n];

 for (int i=0;i<n;i++)

 s[i]=new int[n];

 init(n,s);

 skoczek(n,1,0,0,s);

 system("pause");

}

Listing 11.5 Tekst programu rozwiązujący problem skoczków

Wynik działania programu został pokazany na rys. 11.6, rysunek pokazuje początkowe rozwiązania, wszystkich rozwiązań jest 304.



Rys. 11.6. Przykład działania programu, którego tekst został pokazany na rys. 11.5 dla n=5

11.5. Zadania na Laboratorium Programowania

1. Dana jest tablica jednowymiarowa zawierająca n-elementów. Wygeneruj losowo jej elementy za pomocą funkcji, n jest liczbą losową z <10,15>. Wypisz elementy w odwrotnej kolejności względem indeksu (najpierw elementy o indeksie największym) o wartościach podzielnych przez -3 i -4. Użyj rekurencji.
2. Dana jest tablica jednowymiarowa zawierająca n-elementów. Wygeneruj losowo jej elementy za pomocą funkcji, n jest liczbą losową z <-18,-3>. Wypisz te elementy o indeksach parzystych, które są podzielne przez -7. Użyj rekurencji.
3. Dana jest tablica jednowymiarowa zawierająca n-elementów. Wygeneruj losowo jej elementy za pomocą funkcji, n jest liczbą losową z <-4,13>. Wypisz te elementy o indeksach nieparzystych, które są podzielne przez -6. Użyj rekurencji.
4. Napisz funkcję rekurencyjną obliczającą wartość funkcji cos(x) z dokładnością eps. Zdefiniuj funkcję z parametrami: (float x, float eps=0.0001). Wartość funkcji oblicz jako sumę szeregu: W obliczeniach kolejnych wyrazów szeregu należy korzystać wyłącznie z operatorów arytmetycznych. Napisz program wywołujący tą funkcję.
5. Napisz funkcję rekurencyjną obliczającą wartość funkcji sin(x) z dokładnością eps. Zdefiniuj funkcję z parametrami: (float x, float eps=0.0001). Wartość funkcji oblicz jako sumę szeregu: W obliczeniach kolejnych wyrazów szeregu należy korzystać wyłącznie z operatorów arytmetycznych. Napisz program wywołujący tą funkcję.
6. Napisz funkcję rekurencyjną obliczającą wartość funkcji log(x) z dokładnością eps. Zdefiniuj funkcję z parametrami: (float x, float eps=0.0001). Napisz program wywołujący tą funkcję.
Wartość funkcji oblicz jako sumę szeregu: w obliczeniach kolejnych wyrazów szeregu należy korzystać wyłącznie z operatorów arytmetycznych. Napisz program wywołujący tą funkcję.
7. Dana jest tablica dwuwymiarowa kwadratowa o wymiarach NxN. Napisz program, który wypełni liczbami losowymi rzeczywistymi lewą górną część tablicy a następnie przekopiuje wartości do prawej dolnej. Wypisz tablicę z danymi początkowymi i końcowymi, użyj formatu z dwoma cyframi po kropce. Użyj rekurencji.
8. Dana jest tablica dwuwymiarowa kwadratowa o wymiarach NxN. Napisz program, który wypełni liczbami losowymi rzeczywistymi lewą dolną część tablicy a następnie przekopiuje wartości do prawej górnej. Wypisz tablicę z danymi początkowymi i końcowymi, użyj formatu z dwoma cyframi po kropce. Użyj rekurencji.
9. Dana jest tablica dwuwymiarowa kwadratowa o wymiarach NxN, N>5. Napisz program, który wypełni liczbami losowymi rzeczywistymi główną przekątną oraz dwie sąsiednie z lewej i z prawej strony (jest to macierz trój diagonalna). Wypisz tablicę z danymi początkowymi i końcowymi, użyj formatu z dwoma cyframi po kropce. Użyj rekurencji.

11.6. Zadania do samodzielnego rozwiązania

1. Napisz funkcję rekurencyjną czytającą n – liczb całkowitych, n – podane parametrem, oraz wypisującą podane liczby:
	1. w porządku odwrotnym do czytania danych,
	2. zgodnie z porządkiem podawania danych.

Wywołaj tę funkcjeę w funkcji main.

WEJŚCIE

5, 12, 34, 1, 3

WYJŚCIE

a) 3, 1, 34, 12, 5 b) 5, 12, 34, 1, 3

1. Napisz funkcję rekurencyjną obliczającą dla podanej parametrem liczby jej wartość pisaną wspak.

Wywołaj tę funkcję w funkcji main dla podanej parametrem liczby.

1. Zdefiniuj funkcję rekurencyjną obliczająca sumę n-elementów szeregu:

1+1/2 + 1/3 + ¼ + 1/5 + ….1/n

WEJŚCIE

n=5

WYJŚCIE

2.283(3)

1. Zdefiniuj funkcję rekurencyjną NWW, obliczającą najmniejszą wspólna wielokrotność dwu liczb naturalnych podanych jako parametry funkcji.
Wykorzystaj tę funkcję do obliczenia NWW podanych przez użytkownika liczb.

Do obliczenia NWW wykorzystaj metodę rozkładu na czynniki pierwsze.

WEJŚCIE

a=300, b=700

WYJŚCIE

2100

1. Zdefiniuj funkcję rekurencyjną funkcje obliczająca wartość funkcji Ackermana



Wykonaj obliczenia dla m<=3 i n<=4.

WEJŚCIE

m=3, n=4

WYJŚCIE

125

1. Napisz funkcję rekurencyjną dzielniki (int, int) wypisującą rozkład liczby podanej parametrem na czynniki pierwsze. Czynniki pierwsze to liczby pierwsze, które są dzielnikami liczby. Wykorzystaj funkcję w programie, w którym w funkcji main wczytana jest liczba i wywołana funkcja dzielniki.

11.7. Zadania z podstaw programowania

1. Dany jest wzór:

$$a\_{n}= \left\{\begin{array}{c}1, gdy n=0\\2, gdy n=1\\a\_{n-1}\*a\_{n-2}, gdy n>1\end{array}\right.$$

Oblicz an. Użyj rekurencji.

1. Dany jest wzór:

$$a\_{n}= \left\{\begin{array}{c}-1, gdy n=0\\2, gdy n=1\\a\_{n-1}-a\_{n-2}+5, gdy n>1\end{array}\right.$$

Oblicz an. Użyj rekurencji.

1. Dany jest wzór:

$$a\_{n}= \left\{\begin{array}{c}1, gdy n=0\\2, gdy n=1\\a\_{n-1}+a\_{n-2}-4, gdy n>1\end{array}\right.$$

Oblicz an. Użyj rekurencji.

1. Dany jest ciąg n-elementowy liczb rzeczywistych losowych z przedziału <-3,9). Napisz program obliczający maksimum dla tego ciągu, użyj formatu z dwoma cyframi po kropce. Wypisz dane i wyniki, , użyj formatu z dwoma cyframi po kropce. Użyj rekurencji.
2. Dany jest ciąg n-elementowy liczb rzeczywistych losowych z przedziału <2,18). Napisz program obliczający iloczyn elementów dla tego ciągu. Wypisz dane i wyniki, , użyj formatu z dwoma cyframi po kropce. Użyj rekurencji.
3. Dany jest ciąg zakończony zerem liczb rzeczywistych losowych <-2,8). Napisz program obliczający sumę elementów dla tego ciągu. Wypisz dane i wyniki, , użyj formatu z dwoma cyframi po kropce. Użyj rekurencji.
4. Dana jest tablica n-elementowa liczb całkowitych. Napisz program wstawiający nowy element w miejsce o indeksie k. Funkcja wczytująca dane, wstawiająca i wypisująca wykorzystują rekurencję. Wypisz dane przed i po wstawieniu elementu.
5. Dana jest tablica n-elementowa liczb całkowitych zapisanych w układzie dziesiętnym. Napisz program, który przekształca te liczby na liczby zapisane w układzie dwójkowym. Wypisuj je w sposób naturalny, tzn. od lewej wypisywane są cyfry najbardziej znaczące. Użyj rekurencji.
6. Dana jest tablica n-elementowa liczb całkowitych. Napisz program obliczający sumę cyfr tych liczb. Wypisz dane i wynik obliczeń. We wczytywaniu danych oraz obliczeniach użyj funkcji rekurencyjnych.