



Część 6

Funkcje, Rekurencja

Własne funkcje

Ćwiczenia wykonane pisemnie (notatki w zeszycie)

Zapisz numer pytania (przed pytaniem np. Pytanie 1) poniżej zapisz treść pytania. Treść pytania podkreśl na **zielono** a pod treścią pytania zapisz odpowiedź.

Pytanie 1

Opisz, jaki jest cel stosowania funkcji.

Pytanie 2

Opisz jak wygląda składnia funkcji.

Pytanie 3

Opisz pojęcie zestawu parametrów funkcji.

Pytanie 4

Co zawierają treść funkcji?

Pytanie 5

Opisz instrukcję return → z przykładem funkcji o dwóch return

Pytanie 6

Podaj, w jaki sposób wywołujemy funkcję.

Pytanie 7

Zanotuj, na czym polega przekazywanie argumentów funkcji przez wartości oraz przez referencję.

Pytanie 8

Opisz zasadność przekazywanie argumentów funkcji przez wartości oraz przez referencję.

Pytanie 9

Jak definiujemy i jak działają zmienne lokalne i globalne?

Pytanie 10

Co to jest programowanie proceduralne?

Pytanie 11

Opisz budowę funkcji.

Pytanie 12

Podaj zalety i wady stosowania funkcji.

Pytanie 13

Jaki typ funkcji nazywamy procedurą?

Własne funkcje

Definiowanie funkcji

Funkcja = to tak naprawdę podprogram;
czyli jakaś konkretna czynność do wykonania,
podczas gdy główny program - `int main()` - służy jedynie do zarządzania funkcjami

reasumując:

Funkcja – podprogram mający za zadanie wykonać pewne działania nadanych wejściowych i zwrócić głównemu programowi wynik swoich działań.

Cel stosowania funkcji.

- Część programu zdefiniowana, jako funkcja może być wykorzystywana w wielu miejscach programu, co zmniejsza wielkość kodu całego programu.
- Program napisany z użyciem funkcji jest czytelniejszy łatwiejszy do analizowania.

Budowa funkcji

nagłówek funkcji

```
{
ciało funkcji;
}
```

Składnia funkcji.

```
typ_wyniku nazwa_funkcji (zestaw parametrów)
{
    treść funkcji;
}
```

typ_wyniku - jaki będzie zwracała funkcja;

Nazwa_funkcji – tutaj nadajemy nazwę funkcji (brak polskich znaków, spacji i nie zaczynanie się od cyfry);

() – nawiasy okrągłe są zarezerwowane dla funkcji;

zestaw parametrów (argumentów) → są to nazwy obiektów przekazywanych do wnętrza funkcji wraz z ich typem, oddzielone przecinkami, parametry są w nawisach okrągłych, gdy funkcja nie ma parametrów to są puste nawiasy.

treść funkcji → to instrukcje i operacje wykonywane przez funkcję.

instrukcja powrotu return.

Służy do zakończenia wykonania zawierającej ją funkcji i może mieć postać:

```
return;
return stała;
return Wyrażenie;
return (wyrażenie);
wewnątrz funkcji mogą być dwa polecenia
return
np.
```

```
float wartosc_bezwzględna(float x)
{
    if (x>=0)
        return x;
    if (x<0)
        return -x;
}
```

Wyobraźmy sobie taki przykład:

Definiujemy funkcję dodaj() zwracającą, poprzez instrukcję **return** wartość przekazanego jej w momencie wywołania argumentu powiększoną o 5.

```
float dodaj(float x)
{
    x+=5;
    return x;
}
```

Funkcja dodaj() zwraca wartość i nadaje tę wartość zmiennej liczba zadeklarowanej na zewnątrz funkcji i znanej w programie głównym. Tym razem funkcję umieścimy przed funkcją główną int main(). A oto program w całości:

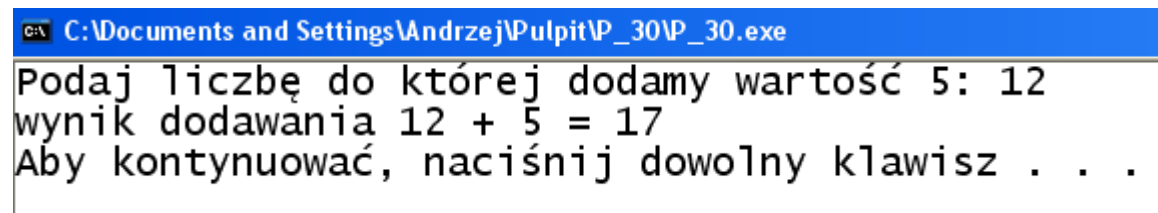
Przykład 30

```
#include <cstdlib>
#include <iostream>

using namespace std;

float liczba;
float dodaj(float x) // tutaj umieszczamy funkcję dodaj
{
    x+=5;           // zmienna x jest zmienną lokalną wyłącznie na potrzeby funkcji dodaj
    return x;
}

int main(int argc, char *argv[])
{
    cout<<"Podaj liczbę do której dodamy wartość 5: ";
    cin>>liczba;           // zmienna liczba jest zmienną globalną
    cout<<"wynik dodawania "<<liczba<<" + 5 = "<<dodaj(liczba)<<endl; // tutaj wywołuję
    funkcję dodaj z parametrem podanej liczby
    system("PAUSE");
    return EXIT_SUCCESS;
}
```



```
C:\Documents and Settings\Andrzej\Pulpit\p_30\p_30.exe
Podaj liczbę do której dodamy wartość 5: 12
wynik dodawania 12 + 5 = 17
Aby kontynuować, naciśnij dowolny klawisz . . .
```

ZADANIE 59

Na podstawie przykładu 30 napisz program, który użyje funkcji z dwoma poleceniami return np. dla przytoczonego wcześniej przykładu na podanie liczby bezwzględnej:

C:\ D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\Zadania\Zadanie_59\Z_30.exe

```
Podaj liczbę: -5
wartość bezwzględna podanej liczby = 5
Aby kontynuować, naciśnij dowolny klawisz . . .
Podaj liczbę: 5
wartość bezwzględna podanej liczby = 5
Aby kontynuować, naciśnij dowolny klawisz . . .
```

wywołanie funkcji:

wywołanie funkcji to napisanie jej nazwy łącznie z nawiasami (w których znajdują się parametry lub wartości, które mają przyjąć parametry).

uwagi:

- nie wolno definiować jednej funkcji wewnątrz innej funkcji;
- nie możesz używać funkcji przed jej zdefiniowaniem;
- **funkcje powinny być** (ze względu na styl programowania i jego czytelność w przypadku długich kodów) **definiowane poniżej funkcji głównej**, gdy chcesz użyć funkcję przed jej zdefiniowaniem musisz **przed funkcją główną napisać instrukcję nagłówka funkcji, którą zdefiniujesz pod funkcją główną** np. `int potega(int a);`

np. w przypadku długiego programu, umieszczenie funkcji `main()` na końcu kodu, spowodowałoby konieczność przewijania całej treści programu co znacząco utrudniałoby nam pracę.

Przykład 31

```
#include <cstdlib>
#include <iostream>

using namespace std;

int potega(int a); //tutaj wypisuję instrukcję nagłówka funkcji potega() zakończoną średnikiem
int liczba;

int main(int argc, char *argv[])
{
    system("color f0");
    cout<<"Podaj liczbę: ";
    cin>>liczba;
    cout<<"Potęga liczby "<<liczba<<" = "<<potega(liczba)<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
int potega(int a) // zdefiniowanie funkcji potega()
{
    return a*a;
}
```

C:\ D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\Przyklady\P_31\P_31.exe

```
Podaj liczbę: 6
Potęga liczby 6 = 36
Aby kontynuować, naciśnij dowolny klawisz . . .
```

- funkcja która nie przypisuje żadnej wartości jest typu pustego czyli **void** - nie musi zawierać polecenia return.

np.

```
void gwiazdki (void)
{
    cout<<"*****";
}
```

Inaczej taki program możemy nazwać:

Procedurą – podprogram, który nie zwraca żadnej wartości funkcji wywołującej.

Zmienne lokalne → zmienne, które są definiowane wewnątrz funkcji, zmienne nie są widoczne poza funkcją.

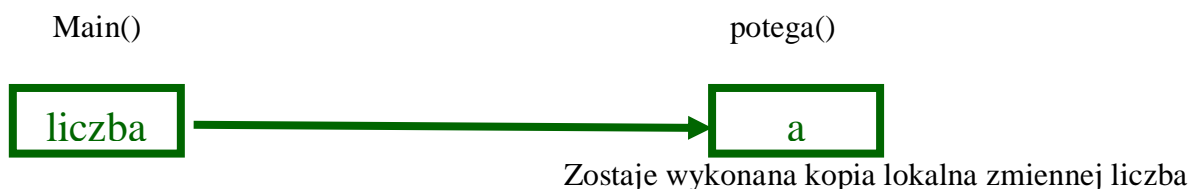
Zmienne globalne → zmienne zdefiniowane poza funkcją main oraz poza innymi funkcjami są widoczne we wszystkich funkcjach programu.

Programowanie proceduralne paradygmat programowania zalecający dzielenie kodu na podprogramy, czyli fragmenty wykonujące ściśle określone operacje, gdzie program komputerowy to zespół współpracujących ze sobą podprogramów, zarządzanych programem głównym o nazwie main().

Przekazywanie argumentów funkcji przez **wartości** oraz przez **referencję**.

Przekazywanie przez wartości → W przypadku wywołania funkcji z przekazywaniem argumentu przez wartość, wykonywana jest kopia lokalna tego argumentu. Wywołana funkcja wykonuje swoje operacje na utworzonej kopii. Po zakończeniu działania funkcji, kopia lokalna jest niszczone.

W przykładzie 31, funkcja główna main() operuje na zmiennej liczba, zaś funkcja potega() operuje na zmiennej lokalnej a:



W momencie wywołania funkcji potega() tworzona jest na jej potrzeby zmienna lokalna o nazwie **a** i do niej jest kopiowana wartość zmiennej **liczba** wpisaną z klawiatury i przekazaną do funkcji potega().

Funkcja `potega()` wykonuje więc, jedynie działanie podnoszenia do potęgi zmiennej `a` i nie zajmuje się niczym innym.

Po zakończeniu działania funkcji `potega()` zmienna `a` powiązana z parametrami przekazywanymi do funkcji przestaje istnieć. Po wyjściu z funkcji `potega()` znów odwołujemy się do oryginalnej zmiennej `liczba`, która nie została zmodyfikowana.

Dzieje się tak dlatego że funkcja główna nie widzi co się dzieje ze zmienną lokalną `a`, jakie działania wykonał na niej podprogram. Dla uniknięcia błędów funkcja główna sama nadzoruje wartość zmiennej `liczba` i nie udostępnia jej w orginale podprogramowi.

Przesyłanie przez wartość – podprogram otrzymuje kopię zmiennej!

Przekazywanie przez referencję → W momencie wywołania wszelkie operacje są wykonywane na oryginalnych zmiennych. Po zakończeniu działania funkcji odwołujemy się do zmiennej zmodyfikowanej. Aby zaznaczyć, że chodzi o referencje przed nazwą zmiennej wpisujemy znak `&`.



Przesyłanie przez referencję – podprogram otrzymuje orginał zmiennej!

Do czego jest potrzebne przekazywanie argumentów funkcji przez wartości oraz przez referencję?

- 1) Przekazywanie parametrów przez wartość stosuje się wtedy, gdy funkcja ma na ich podstawie jedynie coś wyliczyć.
- 2) Przekazywanie parametrów przez referencję, czyli poprzez adres zmiennej stosujemy wtedy, gdy funkcja powinna coś w tej zmiennej umieścić, np. wynik obliczeń. W ten sposób pojedyncza funkcja może zwrócić wiele różnych wyników w kilku zmiennych, których adresy otrzymała na liście parametrów.
- 3) Referencja jest wygodna także wtedy, gdy przekazywany obiekt (zmienna) ma duży rozmiar.
- 4) Przekazanie przez wartość zawsze wymaga tworzenia kopii obiektu (zmienna), którą otrzymuje funkcja do swojej wyłącznej dyspozycji. Przy dużych obiektach (zmiennych) prowadzi to do intensywnych operacji pamięciowych, spowalniając działanie programu.
- 5) Referencja natomiast wymaga jedynie przesłania adresu obiektu - zwykle 4 bajty. Jest zatem bardzo szybka. Jednakże z uwagi na bezpośredni dostęp do obiektu, programista musi być bardzo ostrożny, aby przypadkowo nie zmienić danych, których funkcja zmieniać nie powinna.

Przykład 32

Wykonaj:

- Przepisz temat zadania.
- Przepisz uwagę.
- Wpisz i uruchom przykład.

Temat:

Przykład wypisuje gwiazdki z zużyciem zdefiniowanej funkcji. Funkcja nie ma, return (czyli nie zwraca swojej wartości) oraz nie ma parametrów.

Uwaga:

Gdy definiujemy funkcję pod funkcją główną (main) to musimy uprzedzić kompilator o tym, że funkcja będzie zdefiniowana później w tym przypadku poprzez:

```
void gwiazdki(void);
```

zapis ten musi być nad słowem kluczowym main.

```
#include <cstdlib>
#include <iostream>

using namespace std;
void gwiazdki(void); // uprzedzam kompilator -> funkcja gwiazdki będzie później

int main(int argc, char *argv[])
{
    gwiazdki();
    cout<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

```
void gwiazdki (void)
```

```
{
    cout<<"*****"; /* funkcja która nie przypisuje żadnej wartości jest typu
    pustego czyli void - nie musi zawierać polecenia return.*/
}
```

```
C:\ D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\PrzykladyP_32P_32.exe
```

```
*****
```

```
Aby kontynuować, naciśnij dowolny klawisz . . . _
```

ZADANIE 60

Wykonaj:

- Przepisz temat zadania.
- Rozwiąż zadanie.
- Wpisz listing do zeszytu.

Temat: Napisać program z użyciem procedur wyświetlający na monitorze trzy prostokąty oraz trzy trójkąty konstruowane z znaków "*".

Zdefiniuj procedury:

```
PROSTOKAT_nazwisko_ucznia
```

```
TROJKAT_nazwisko_ucznia
```

Następnie wywołaj trzy razy te procedury. Drukuj trójkąty i prostokąty naprzemian.

Zalety stosowania funkcji:

- nie powtarzamy tego samego kodu kilka razy i skracamy długość kodu, czyli unikamy redundancji;
- szybsza i łatwiejsza poprawa błędów;
- struktura programu jest poukładana i ma sens;
- podział prac pod względem funkcjonalności (w przypadku pracy w zespole programistów) ;
- oszczędność czasu, dzięki wykorzystaniu raz stworzonych funkcji;
- łatwiejsza lokalizacja błędów podczas testów aplikacji;
- lepsze wykorzystanie pamięci komputera.

Wada stosowania funkcji:

- poświęcenie czasu na zrozumienie działania funkcji.

ZADANIE 61

Uruchom dwa przykłady umieszczone poniżej (Przykład pierwszy, Przykład drugi) jako osobne programy.

Zanotuj w zeszycie wyniki działania obu przykładów poprzez przepisanie wyniku działania (wygląd ekranu), podpisz, który wygląd, czego dotyczy oraz czym się różnią. Dodaj komentarze do odpowiednich linijek kodu.

Przykład pierwszy

```
//przekazywanie przez wartości
#include <cstdlib>
#include <iostream>

using namespace std;
void przekaz(int x);
int main(int argc, char *argv[])
{
    int x=10;
    przekaz(x);
    cout<<"\n";
    cout<<"na zewnątrz funkcji x="<<x<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
void przekaz (int x)
{
    x = 2*x;
    cout<<"wewnątrz funkcji x="<<x<<endl;
}
```


Przykład drugi

```
//przekazywanie przez referencję
#include <cstdlib>
#include <iostream>

using namespace std;
void przekaz(int &x);
int main(int argc, char *argv[])
{
    int x=10;
    przekaz(x);
    cout<<"\n";
    cout<<"na zewnątrz funkcji x="<<x<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}
void przekaz (int &x)
{
    x = 2*x;
    cout<<"wewnątrz funkcji x="<<x<<endl;
}
```

Przykład 33

Wykonaj:

- Przepisz temat zadania.
- Przepisz do zeszytu, co to jest tablicowanie funkcji oraz uzupełnij napisy „uzupełnij sam”
- Wpisz i uruchom przykład.

Temat: Tablicowanie funkcji

Tablicowanie funkcji polega na wypisaniu wartości funkcji dla określonego przedziału argumentów funkcji z zadanyim krokiem.

Np. dla $y=x*x*x-2$ z krokiem 1 z przedziału <2,10>

f(2)=6

f(3)=25

f(4)=62

f(uzupełnij sam)=uzupełni sam

.....

Treść przykładu.

Program tablicujący funkcję $y=x*x$ z przedziału od 5 do 15 z krokiem 0,1.

Z wykorzystaniem definiowania funkcji.

```
#include <stdlib.h>
#include <iostream.h>
#include <math.h>
#include <conio.h>

//definiowanie funkcji
```

```

float f(float x)
{
    return x*x;
}

//rozpoczecie programu glownego
using namespace std;
int main(int argc, char *argv[])
{
    float x=5;
    int z=0; // zmienna zliczajaca ilosc wierszy
    for(;x<=15;x=x+0.1)
    {
        z++;
        cout.precision(3); //określenie miejsca po przecinku --> jedno
        cout<<"f("<<x<<"="; // tworzymy napisy f(x)
        cout.precision(4); //określenie miejsca po przecinku --> dwa
        cout<<f(x)<<"\n"; //wypisujemy wartosci funkcji f(x)
        if (z%22==0) //druk co 22 wiersze i pauza
            //poprzez sprawdzenie podzielności licznika wierszy --> z przez 22
            {
                system("PAUSE");
            }
    }
    system("PAUSE");
    return EXIT_SUCCESS;
}

```

W tym przykładzie podczas kompilacji może się pojawić następujący komunikat:

„#warning This file includes at least one deprecated or antiquated header. Please consider using one of the 32 headers found in section 17.4.1.2 of the C++ standard. Examples include substituting the <X> header for the <X.h> header for C++ includes, or <iostream> instead of the deprecated header <iostream.h> “. Komunikat ten informuje nas o potrzebie dołączenia nowej <iostream> biblioteki zamiast starej <iostream.h>.

ZADANIE 62

Stablicować funkcję z krokiem 0,1 przy wykorzystaniu definiowania funkcji z zatrzymywaniem ekranu po 15 wyświetleniach. Nazwa funkcji f_nazwisko_ucznia, nazwa argumentu x_nazwisko_ucznia.

$$f(x) = \begin{cases} x^2 - 12x + 36 & \text{dla } <5,6> \\ 1 & \text{dla } <\frac{\pi}{4}, 5> \\ \operatorname{tg}(x) & \text{dla } <0, \frac{\pi}{4}> \end{cases}$$

```

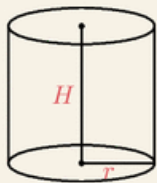
D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\Zadania\Zadanie_62\Z_62.exe
f(0)=0
f(0.1)=0.1003
f(0.2)=0.2027
f(0.3)=0.3093
f(0.4)=0.4228
f(0.5)=0.5463
f(0.6)=0.6841
f(0.7)=0.8423
f(0.8)=1
f(0.9)=1
f(1)=1
f(1.1)=1
f(1.2)=1
f(1.3)=1
f(1.4)=1
Aby kontynuować, naciśnij dowolny klawisz . . .

```

Przykład 34

Napisać program obliczający pole powierzchni całkowitej P_c oraz objętość walca V . W tym celu zdefiniuj funkcje P_c oraz V ze zmiennymi H i R . Oblicz pole oraz objętość dla

- $R=5$ oraz $H=10$ (ściśle określone)
- dla dowolnych wczytanych R i H wczytanych z klawiatury



H – wysokość walca
 r – promień podstawy (koła)

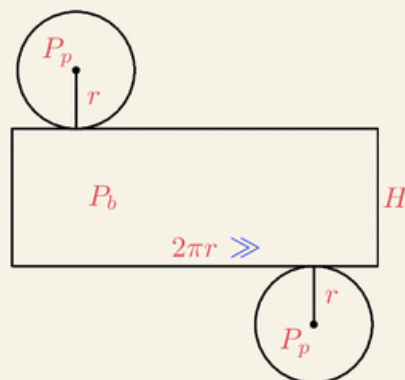
Objętość walca: $V = \pi r^2 H$

Powierzchnię walca można rozciąć na dwa koła i prostokąt o bokach $2\pi r$ i H

Pole podstaw walca: $P_p = \pi r^2 \gg$

Pole powierzchni bocznej: $P_b = 2\pi r H \gg$

Pole całkowite: $P_c = 2P_p + P_b = 2\pi r^2 + 2\pi r H$



```

#include <cstdlib>
#include <iostream>
#include <math.h>
#define H 10.0
#define R 5.0

```

```

float v(float h,float r)
{
    return M_PI*r*r*h;
}
float Pc(float h,float r)
{
    return(2*M_PI*r*r+2*M_PI*r*h);
}

using namespace std;

int main(int argc, char *argv[])
{
    cout<<"H="<<H<<"cm\tr="<<R<<"cm\n";
    cout<<"V("<<H<<","<<R<<")=\t"<<v(H,R)<<"cm^3\n";
    cout<<"Pc("<<H<<","<<R<<")=\t"<<Pc(H,R)<<"cm^2\n";
    float h,r;
    cout<<"Podaj r=";
    cin>>r;
    cout<<"Podaj H=";
    cin>>h;
    cout<<"V("<<h<<","<<r<<")=\t"<<v(h,r)<<"cm^3\n";
    cout<<"Pc("<<h<<","<<r<<")=\t"<<Pc(h,r)<<"cm^2\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}

```

```

D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\Przyklady\p_34\p_34.exe
H=10cm  r=5cm
V(10,5)=          785.398cm^3
Pc(10,5)=          471.239cm^2
Podaj r=12
Podaj H=24
V(24,12)=          10857.3cm^3
Pc(24,12)=          2714.34cm^2
Aby kontynuować, naciśnij dowolny klawisz . . .

```

ZADANIE 63

Napisać program obliczający pole powierzchni oraz obwód prostokąta.

Zdefiniuj funkcje:

POW_nazwisko_ucznia

OBW_nazwisko_ucznia.

Wywołaj funkcje POW oraz OBW z A=4 i B=5. Wypisz na ekranie wyniki w odpowiednim formacie oraz z jednostkami. Napisz tak program ,aby komputer pytał się o A i B następnie liczył POW i OBW dla różnych A i B. Wypisz na ekranie wyniki w odpowiednim formacie oraz z jednostkami.

```

C:\ D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\Zadania\Zadanie_63\Z_63.exe
A=4cm    B=5cm
POW(4,5)= 20cm^2
OBW(4,5)= 18cm
Podaj A=2
Podaj B=7
POW(2,7)= 14cm^2
OBW(2,7)= 18cm
Aby kontynuować, naciśnij dowolny klawisz . . . _

```

Przykład 35

Temat: Funkcja typu bool określająca czy wczytana liczba jest pierwsza.

Algorytm:

Określamy ilość podzielników wczytanej liczby. Jeśli ilość podzielników jest 2 to liczba jest pierwsza wtedy funkcja przyjmuje wartość true w przeciwny przypadku to false.

```

#include <cstdlib>
#include <iostream>

using namespace std;
bool pierwsza(int liczba)
{
    int ilosc=0;
    for (int i=1;i<=liczba;i++)
    {
        if(liczba%i==0) /* sprawdzamy wszystkie podzielniki w tym 1 oraz samą siebie -
liczba*/
        { ilosc++; }
    }
    if (ilosc==2) //dwa podzielniki 1 oraz liczba informują nas że liczba jest pierwsza
    { return true; }
    else
    { return false; } //istnieje co najmniej jeszcze jeden podzielnik
}

int main(int argc, char *argv[])
{
    int liczba;
    cout<<"podaj liczbe=";
    cin>>liczba;
    if (pierwsza(liczba)==true)
    { cout<<"liczba jest pierwsza"; }
    else
    { cout<<"liczba nie jest pierwsza"; }
    cout<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}

```

ZADANIE 64

Temat: Program wykorzystujący funkcję z poprzedniego zadania do znajdowania bliźniaczych liczb pierwszych w przedziale od $\langle 1, \text{max} \rangle$ gdzie max jest liczba wczytaną do program.

Bliźniacze liczby pierwsze to liczby pierwsze różniące się o dwa np. 11 13.

```

C:\ D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\Zadania\Zadanie_64\Z_64.exe
podaj górną granicę sprawdzanego przedziału = 45

Znalezione pary liczb bliźniaczych:
3 5
5 7
11 13
17 19
29 31
41 43
Aby kontynuować, naciśnij dowolny klawisz . . .

```

ZADANIE 65

Napisz program obliczający funkcje sumy, różnicy, iloczynu, ilorazu, potęgowania. Wykorzystując stworzone samodzielnie funkcje np.:

```

int suma(int x,int y);
int roznica(int x,int y);
int iloczyn(int x,int y);
float iloraz(int x,int y);
double potega(int x,int y);

```

Sprawdź działanie program dla liczb naturalnych z przedziału $\langle -10;10 \rangle$

```

C:\ D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\
Podaj a:
a = 3
Podaj b:
b = 7

Suma: 3 + 7 = 10
Roznica: 3 - 7 = -4
Iloczyn: 3 * 7 = 21
Iloraz: 3 : 7 = 0.43
Potęga: 3 do potęgi 7 = 2187_

```

ZADANIE 66

Napisz program wyznaczający element minimalny i maksymalny w tablicy a[50], Stwórz funkcje pomocnicze:

```

float minimum(float [],int );
float maksimum(float [],int );

```

```

C:\ D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\
Podaj ilosc elementow ciagu:
n = 5

Wczytaj elementy ciagu:
x[0] = 8
x[1] = -3
x[2] = -44
x[3] = 2
x[4] = 7

Element minimalny ciagu: -44
Element maksymalny ciagu: 8_

```

Rekurencja

Ćwiczenia wykonane pisemnie (notatki w zeszyście)

Zapisz numer pytania (przed pytaniem np. Pytanie 1) poniżej zapisz treść pytania. Treść pytania podkreśl na **zielono** a pod treścią pytania zapisz odpowiedź.

Pytanie 14

Co to jest rekurencja?

Pytanie 15

Co nazywamy przypadkiem podstawowym

Pytanie 16

Podaj definicję silni.

Pytanie 17

Podaj rekurencyjną definicję silni.

Pytanie 18

Podaj rekurencyjną definicję ciągu Fibonaciego. Jakie zastosowania może mieć ciąg Fibonaciego.

Pytanie 19

Podaj, którą metoda jest szybsza rekurencja czy iteracja oraz którą należy kiedy stosować.

REKURENCJA

Rekurencja zachodzi wtedy, gdy jakaś funkcja w celu zwrócenia prawidłowego wyniku **wywołuje samą siebie** (a dokładnie mówiąc: tworzy własne kopie), aż do napotkania tzw. przypadku podstawowego, dla którego funkcja może wyznaczyć wynik.

Przypadek podstawowy występuje dla takiej wartości argumentu funkcji, dla której można już podać wynik (nie trzeba tworzyć kopii).

Rozwiązując zadania rekurencyjne zaczynamy od wymyślenia przypadku podstawowego!

Silnia definicja

W matematyce definiowana jest pojęcie silni np. $n!$

Def:

$0!=1$

$1!=1$

.....

$5!=1*2*3*4*5=120$

Przykład 36

Obliczyć wartość silni stosując rekurencję. Silnia w sposób rekurencyjny jest definiowana:

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n > 0 \end{cases}$$

Wykonaj:

a) sprawdź i zapisz w zeszycie, do jakiej wielkości można obliczać silnię z użyciem tego przykładu.

b) wpisz przykład do zeszytu.

```
#include <cstdlib>
#include <iostream>

double silnia(int n);

using namespace std;

int main(int argc, char *argv[])
{
    int n;
    double wynik;
    cout<<"Obliczam silnie z liczby naturalnej"<<endl;
    cout<<"Podaj liczbe n=";
    cin>>n;
    wynik=silnia(n);
    cout<<"silnia("<<n<<")="<<wynik;
    cout<<endl;
    system("PAUSE");
    return EXIT_SUCCESS;
}

double silnia(int n)
{
    if (n==0)
    {
        return 1;
    }
    else
    {
        return n*silnia(n-1);    // tutaj funkcja silnia wywołuje samą siebie
    }
}
```


ZADANIE 67

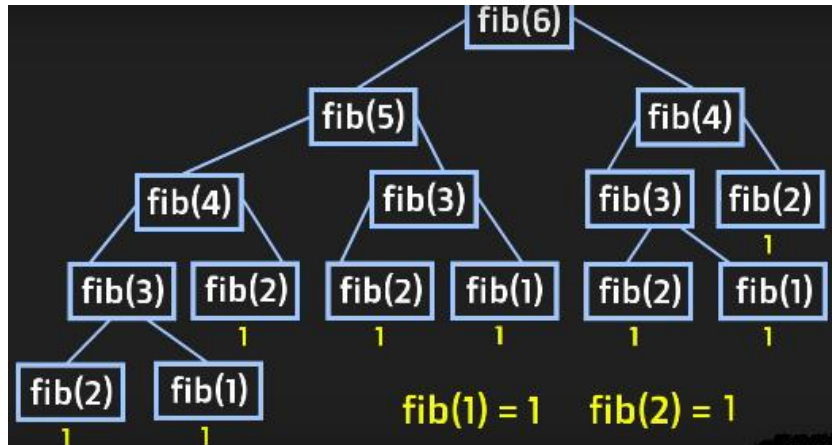
Napisz program rekurencyjnego znajdowania wartości ciągu Fibonacciego. Element ciągu, który mamy obliczyć, wczytujemy z klawiatury.

Przykładowy wygląd ekranu:

Podaj, który element ciągu Fibonacciego, który należy obliczyć = 6

FIBO(6)=8

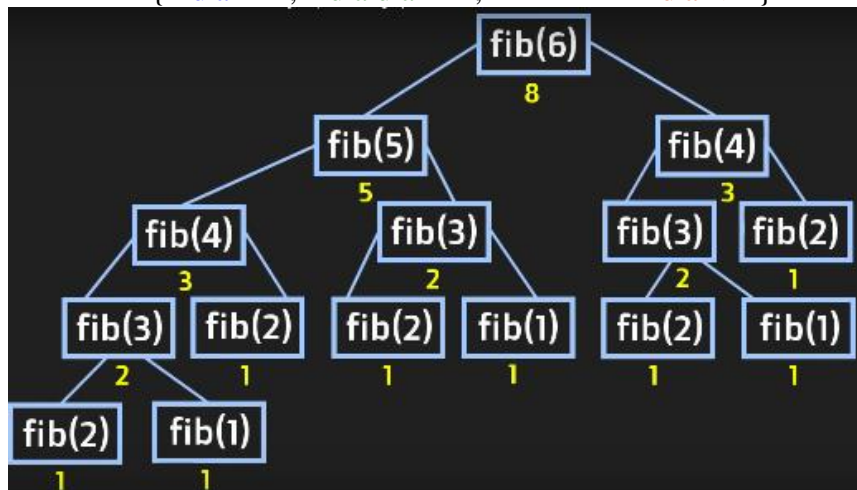
Opis teoretyczny:



Widok drzewa z kopiami funkcji powstającymi aż do momentu uzyskania przypadku podstawowego fib(1)=1 oraz fib(2)=1

Liczby Fibonacciego FIBO(N) pojawiają się przy opisie ilościowym niektórych zjawisk przyrodniczych. Jeśli rozrastanie roślin odbywa się zgodnie z zasadą: każdy pęd wypuściwszy pęd boczny, przez rok odpoczywa i dopiero w następnym roku puszcza nowy pęd, to liczba pędów w N-tym roku może być obliczona zgodnie z wcześniej podanym ciągiem rekurencyjnym FIBO.

$$F_n = \{ 1 \text{ dla } n=1; 1 \text{ dla } n=2; F_{n-2} + F_{n-1} \text{ dla } n>2 \}$$



Widok drzewa z kopiami funkcji powstającymi aż do momentu uzyskania przypadku podstawowego z podstawieniem konkretnych wartości liczbowych np. fib(3)=fib(2)+fib(1)=1+1=2

D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\CZ_6\Zadania\Zadanie_67\Z_67.exe

```

Obliczam wskazany element ciągu Fibonacciego
Podaj, który element ciągu Fibonacciego, który należy obliczyć = 6
FIBO(6)=8
Aby kontynuować, naciśnij dowolny klawisz . . .
  
```

ZADANIE 68

W podpunktach a) b) c) nazwa funkcji to CIAG_nazwisko_ucznia np. CIAG_kowalski
a)

Dany jest ciąg liczbowy $a_1=3$ $a_2=7$ $a_3=16$ $a_4=32$ $a_5=57$ $a_6=93$..

Napisać w zeszycie wzór rekurencyjny obliczający n–ty wyraz ciągu. Napisz na podstawie wzoru rekurencyjnego program obliczający n–ty wyraz ciągu. Oblicz 17–ty wyraz tego ciągu. Zapisz program w zeszycie.

```
C:\Users\andrzej_r\Desktop\C++CZ_6\Zadania\Zadanie_68\Z_68.exe
Obliczam wskazany element ciągu liczbowego
Podaj, element ciągu liczbowego, który należy obliczyć = 17
CIAG(17)=1787
Aby kontynuować, naciśnij dowolny klawisz . . .
```

ZADANIE 69

b)

Dany jest ciąg liczbowy $a_1=2$ $a_2=7$ $a_3=12$ $a_4=17$ $a_5=22$ $a_6=27$..

Napisać w zeszycie wzór rekurencyjny obliczający n–ty wyraz ciągu. Napisz na podstawie wzoru rekurencyjnego program obliczający n–ty wyraz ciągu. Oblicz 20–ty wyraz tego ciągu. Zapisz funkcję rekurencyjną w zeszycie.

```
C:\Users\andrzej_r\Desktop\C++CZ_6\Zadania\Zadanie_69\Z_69.exe
Obliczam wskazany element ciągu liczbowego
Podaj, element ciągu liczbowego, który należy obliczyć = 20
CIAG(20)=97
Aby kontynuować, naciśnij dowolny klawisz . . .
```

ZADANIE 70

c)

Dany jest ciąg liczbowy $a_1=1$ $a_2=9$ $a_3=36$ $a_4=100$ $a_5=225$..

Napisać w zeszycie wzór rekurencyjny obliczający n–ty wyraz ciągu. Napisz na podstawie wzoru rekurencyjnego program obliczający n–ty wyraz ciągu. Oblicz 7–ty wyraz tego ciągu. Zapisz funkcję rekurencyjną w zeszycie.

```
C:\Users\andrzej_r\Desktop\C++CZ_6\Zadania\Zadanie_70\Z_70.exe
Obliczam wskazany element ciągu liczbowego
Podaj, element ciągu liczbowego, który należy obliczyć = 7
CIAG(7)=784
Aby kontynuować, naciśnij dowolny klawisz . . .
```

ZADANIE 71

Temat: Iteracja a rekurencja dwa rozwiązania tego samego problemu.

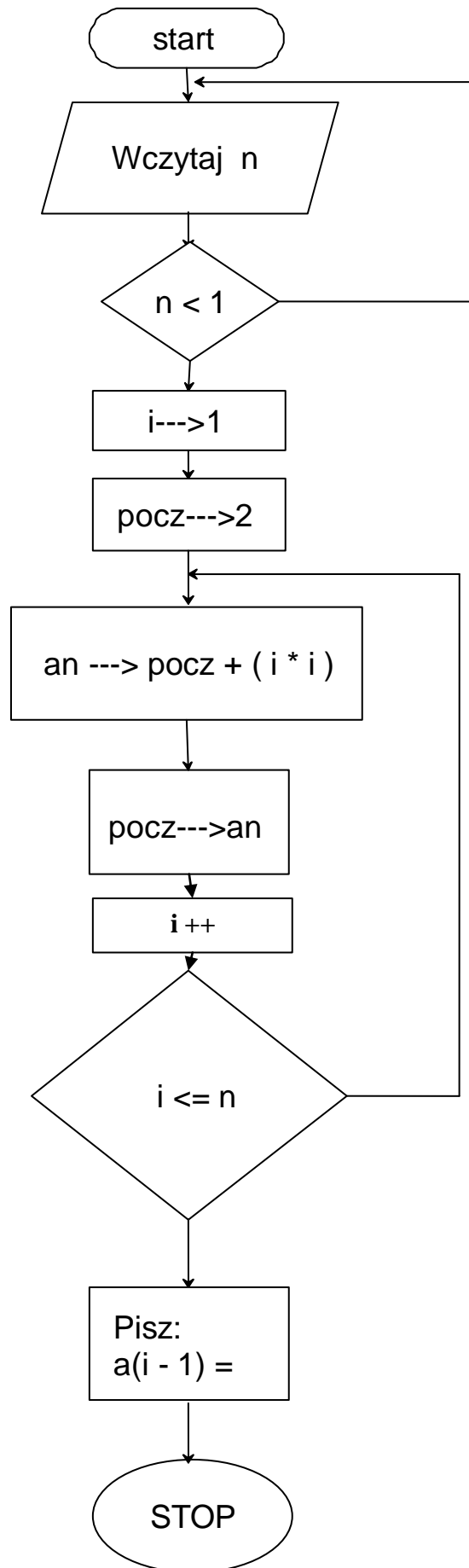
Rozwiąż zadanie numer 68 metodą iteracyjną.

Specyfikacja problemu

Dane wejściowe: Dowolna liczba naturalna n , $n \geq 1$.

Dane wyjściowe: wartość n-tego wyrazu ciągu a_n .

Poniżej przykładowa pętla do...while dla zadania 68



Wykonaj:

- 1) Zanonuj w zeszycie schemat blokowy
- 2) Na podstawie algorytmu napisz program z użyciem pętli do...while.
- 3) zapisz wartości a1, a2, a3, a4, a5, a6.

Algorytmy rekurencyjne charakteryzują się wyższą szybkością działania niż algorytmy iteracyjne. Niestety im więcej tworzonych kopii programu tym bardziej obciążona jest pamięć komputera. W takim przypadku lepiej stosować algorytm iteracyjny.

```
C:\> D:\Andrzej\Praca\Lekcje\CPP\Cwiczenia_C++\C++CZ_6\Zadania\Zadanie_71\Z_71.exe
Podaj, numer elementu ciagu: 6
a(6)=93
Aby kontynuować, naciśnij dowolny klawisz . . . _
```

ZADANIE 72 (Rabaty kwiatowe)

Grażyna bardzo lubi kwiaty. Co roku obsadza rabaty wokół domu różnymi odmianami kwiatów. W tym roku postanowiła, że każdą rabatę o długości k kwiatów obsadzi w inny sposób (tzn. rabaty będą różniły się przynajmniej jednym kwiatem, kwiaty tej samej odmiany mogą się powtarzać). Grażyna ma do dyspozycji n odmian, ilość kwiatów danej odmiany jest nieograniczona. Ile rabat może obsadzić Grażyna? W rozwiązaniu użyj wersji rekurencyjnej funkcji. Przyjmujemy:

n - liczba naturalna ($0 < n < 15$), oznaczająca ilość odmian;

k – liczba naturalna ($0 < k < n$), oznaczająca ilość kwiatów w rabacie.

W wyniku program zwróci nam liczbę będącą ilością różnych rabat, które można obsadzić.

Wyjaśnienie:

Zakładamy że, n=3 (odmiany) oraz k=2 (ilość kwiatów oraz długość rabaty). Mamy więc odmiany 1, 2, 3 długości rabat wynosi 2, co pozwala na stworzenie następującej kombinacji:

1 1, 1 2, 1 3, 2 1, 2 2, 2 3, 3 1, 3 2, 3 3

możemy obsadzić 9 rabat

Widać, że uniknięto tutaj powtarzania układu rabat typu:

1 2, 1 2

1 1, 1 1

3 1, 3 1 itp.

```
C:\> C:\Users\andrzej_r\Desktop\Z_72\Z_72.exe
Pamiętaj aby dane mieściły się w przedziale (0 < k < n)
Podaj ilość odmian n: 1
Podaj ilość kwiatów k: 0
Pamiętaj aby dane mieściły się w przedziale (0 < k < n)
Podaj ilość odmian n: 0
Podaj ilość kwiatów k: 0
Pamiętaj aby dane mieściły się w przedziale (0 < k < n)
Podaj ilość odmian n: 4
Podaj ilość kwiatów k: 3
Ilość rabat możliwych do obsadzenia: 64
Aby kontynuować, naciśnij dowolny klawisz . . .
```

ZADANIE 73 (Liczba permutacji)

Na ile różnych sposobów można ustawić obok siebie n kolejnych liczb naturalnych?

W rozwiązaniu użyj wersji rekurencyjnej funkcji.

Przyjmujemy: n - liczba naturalna ($0 < n < 26$).

Podaj ilość sposobów, na które można ustawić w rzędzie n kolejnych liczb naturalnych

Wyjaśnienie:

Dla $n = 3$ Liczby 1, 2, 3 możemy ustawić na 6 następujących sposobów:

1 2 3

1 3 2

2 1 3

2 3 1

3 1 2

3 2 1

