Informatyka w inżynierii produkcji

Krzysztof Karbowski

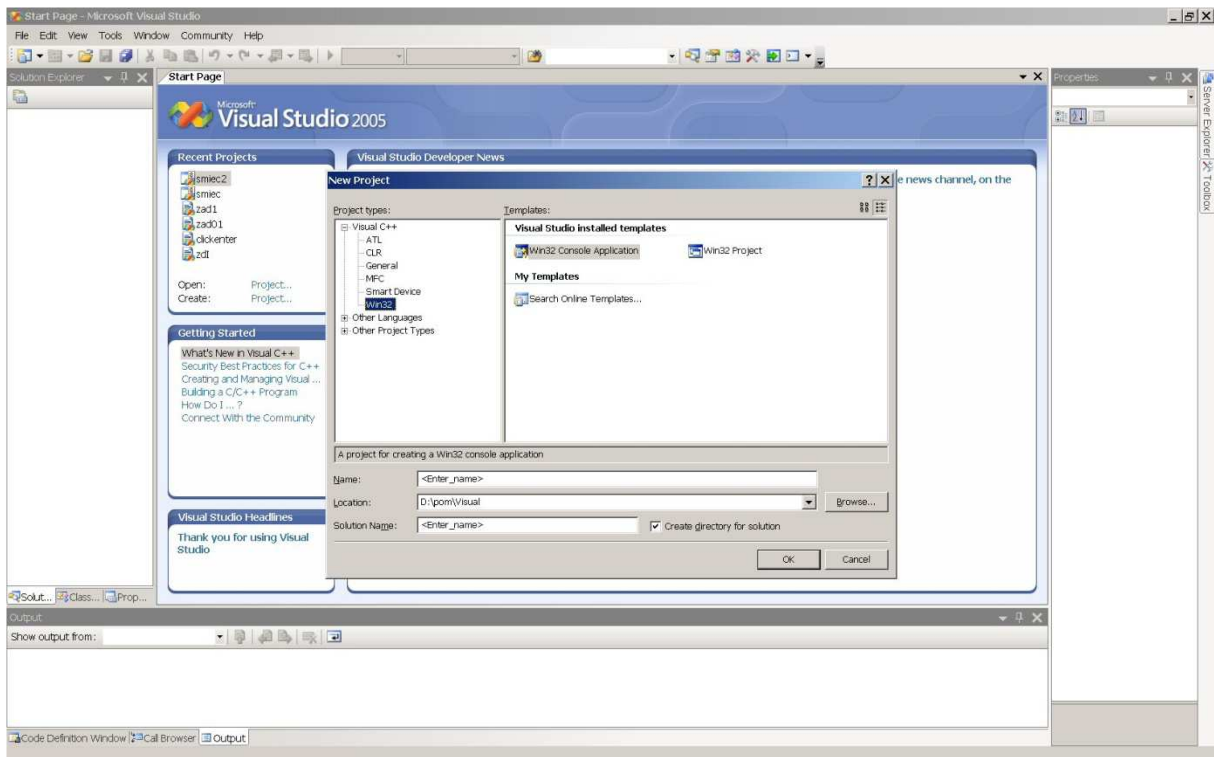
Politechnika Krakowska

Kraków 2015

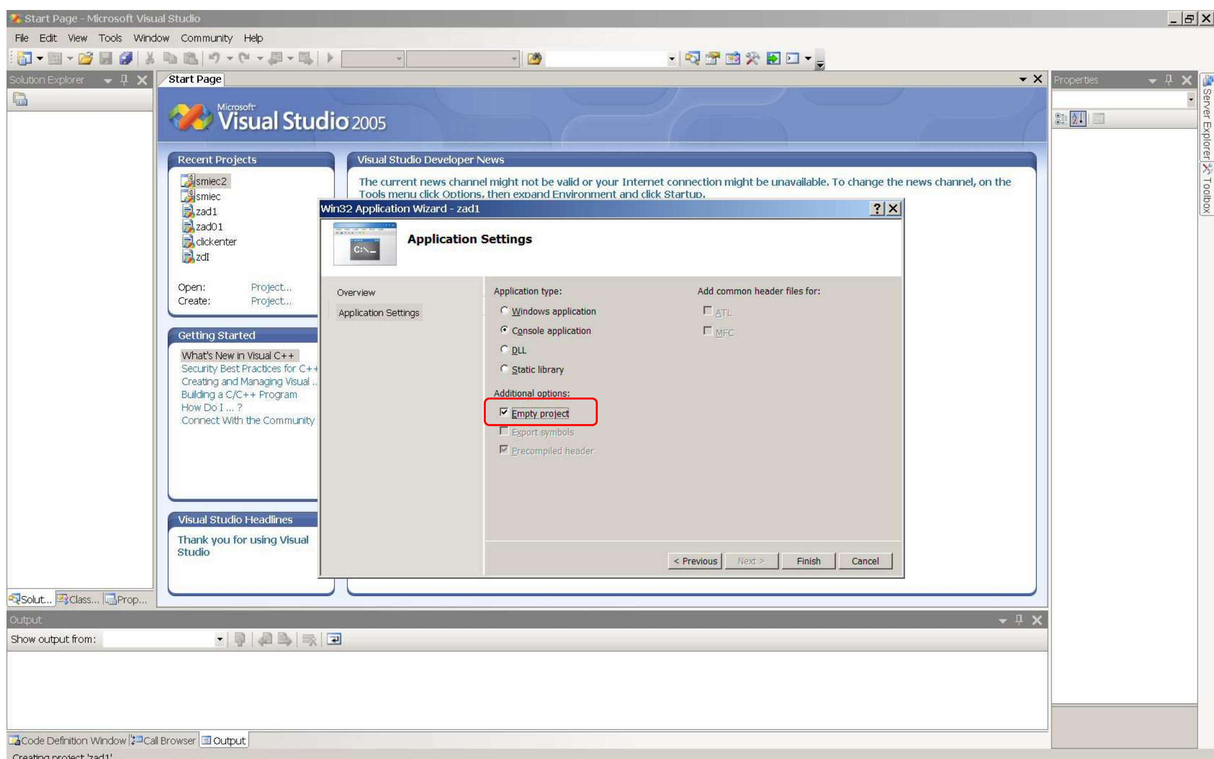
Wydanie: 15.02.2016

Środowisko Visual Studio 2005.

File – New – Project... :

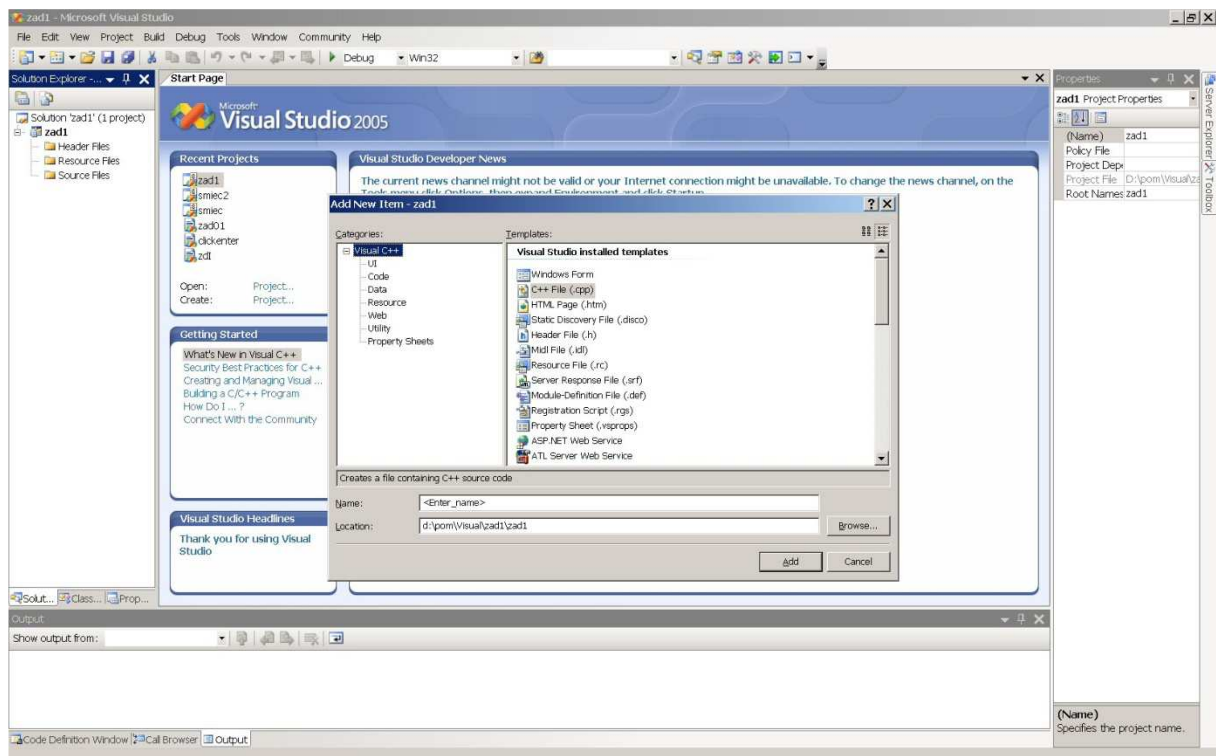


Rys.1.



Rys.2

Project – Add New Item …



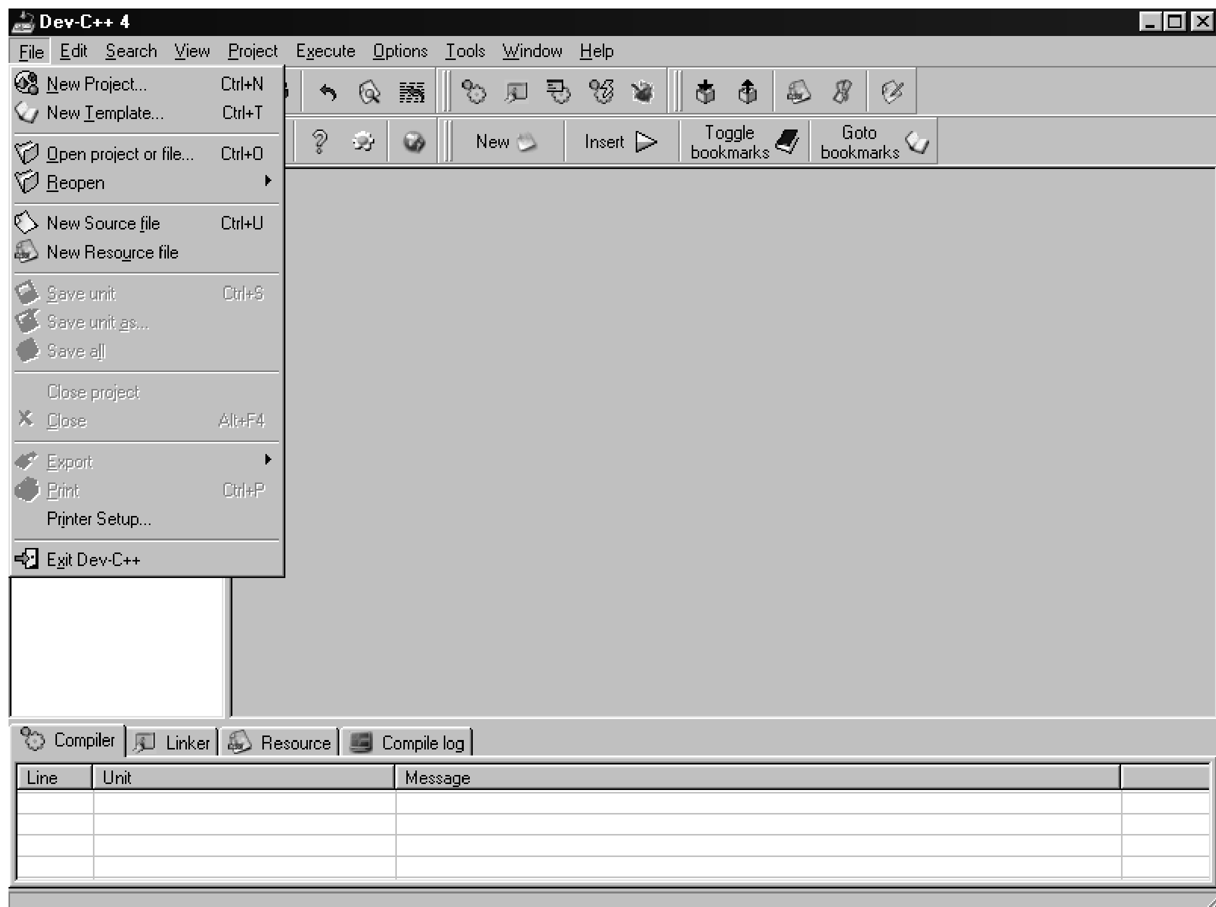
Rys.3.

Kompilacja programu: „Build – Build Solution” (F7).
Uruchomienie programu: "Debug – Debug Without Debugging" (Ctrl+F5)

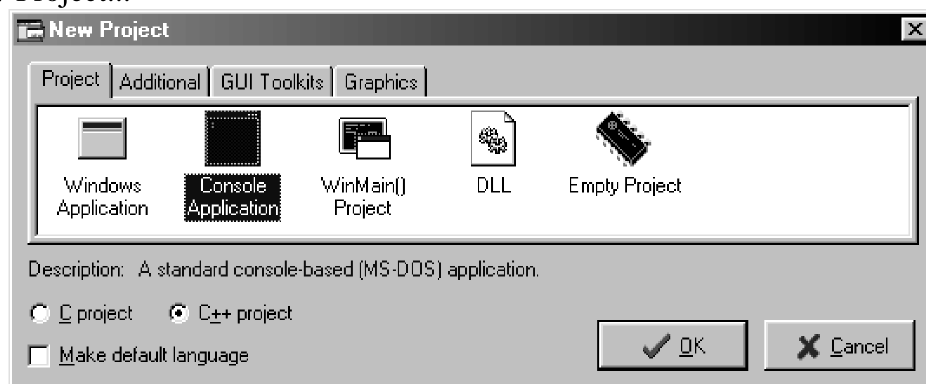
Kompilator Dev-C++

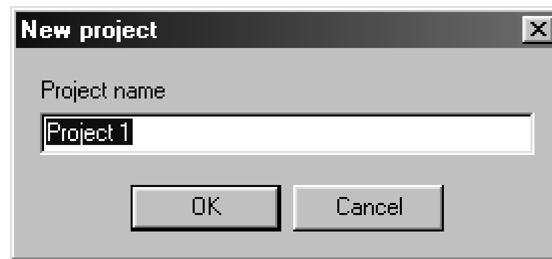
---

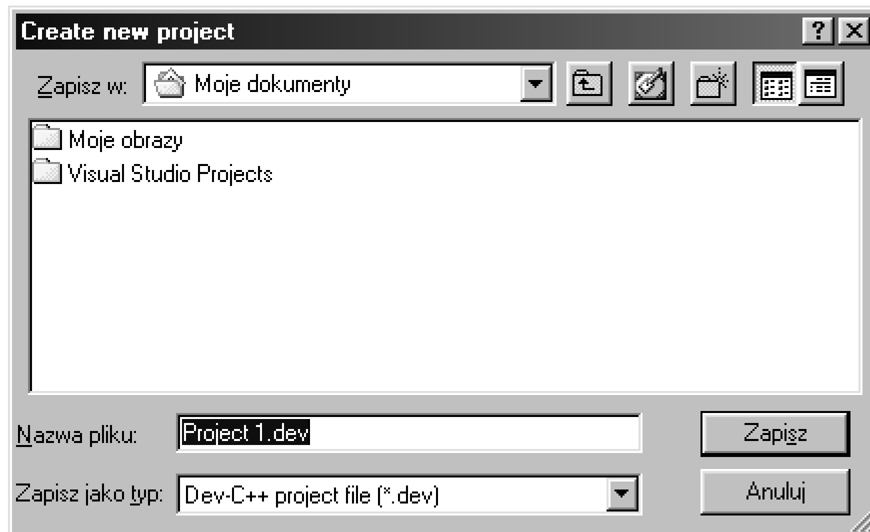
www.bloodshed.net

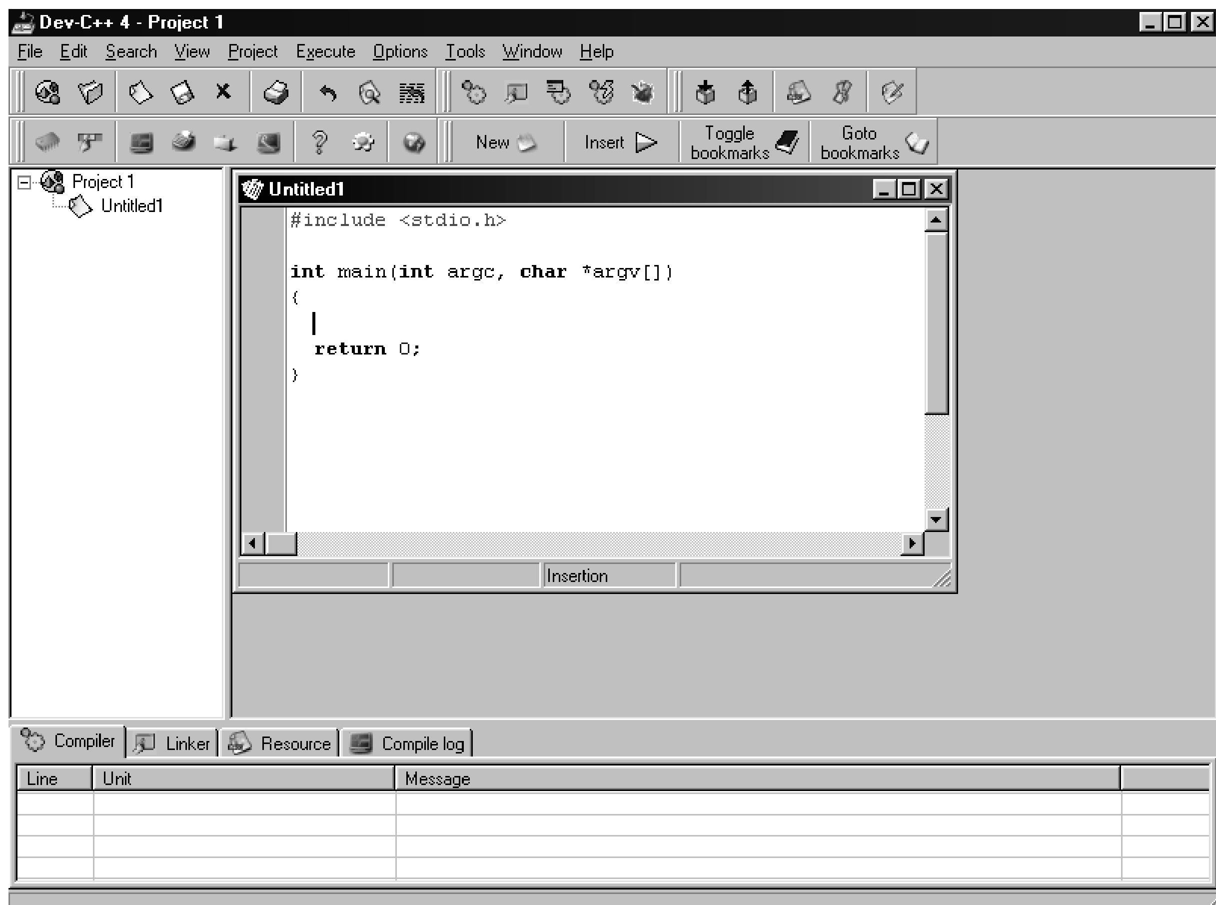


File – New Project...




New project
Project name
Project 1
OK
Cancel


Create new project
Zapisz w:
Moje dokumenty
Moje obrazy
Visual Studio Projects
Nazwa pliku:
Project 1.dev
Zapisz jako typ:
Dev-C++ project file (*.dev)
Zapisz
Anuluj


Dev-C++ 4 - Project 1
File Edit Search View Project Execute Options Tools Window Help
New
Insert
Toggle bookmarks
Goto bookmarks
Project 1
Untitled1
Untitled1
#include <stdio.h>
int main(int argc, char *argv[])
{
return 0;
}
Insertion
Compiler
Linker
Resource
Compile log
Line
Unit
Message

1. Pierwszy program.

wariant 1: ❶

```
#include <iostream>
using namespace std;
int main()
{
   cout << "Pierwszy program \n" ;
   return 0;
}
```

wariant 2: ❷

```
#include <stdio.h>
int main()
{
   printf("Pierwszy program \n") ;
   return 0;
}
```

UWAGA:

C++ jest językiem o wolnym formacie zapisu. Poza nielicznymi wyjątkami instrukcje można zapisywać w dowolnym miejscu. Koniec instrukcji jest oznaczany znakiem „ ; ”.

wariant 3: ❸

```
#include <stdio.h>
main(){ printf("Pierwszy program \n"); return 0;}
```

Kompilacja i konsolidacja programu:

1. kompilator
2. linker

---

2. Drugi program

❹

```
/* ---------------------------------------------------
Program na przeliczanie wysokosci podanej w stopach
na wysokosc w metrach.
Opearacje wejscia/wyjscia.
--------------------------------------------------- */
#include <iostream>
using namespace std;
int main()
{
int stopy;            // definicja zmiennej
float metry;               // definicja zmiennej
float przelicznik=0.3;     // def. i inicjalizacja zmiennej

   cout << "Podaj wysokosc w stopach: " ;
   cin >> stopy ;          // przyjecie danej z klawiatury
   metry = stopy * przelicznik ;
   cout << endl ;
   // wypisanie wynikow
```

```
    cout << stopy << " stop to " << metry
           << " metrow \n" ;
    return 0;
}
```

3. Instrukcje sterujące.

3.1. Prawda – fałsz

wartość zero - odpowiada stanowi: *fałsz (FALSE)*
wartość inna niż zero - odpowiada stanowi: *prawda (TRUE)*

3.2. Instrukcja warunkowa if

Forma 1:

if (*wyrażenie*) instrukcja1 ;

Forma 2:

if (*wyrażenie*) instrukcja1 ;
else instrukcja2 ;

Przykład:

```
int i ;
cout << "Podaj liczbe: " ;
cin >> i ;
if (i - 4) cout << "liczba rozna od 4" ;
else       cout << "liczba rowna 4" ;
```

Wybór wielowariantowy:

if (*wyrażenie1*) instrukcja1 ;
else if (*warunek2*) instrukcja2 ;
else if (*warunek3*) instrukcja3 ;

3.3. Blok instrukcji

{
instrukcja1 ;
instrukcja2 ;
instrukcja*n* ;
}

Przykład:

```
int i ;
cout << "Podaj liczbe: " ;
cin >> i ;
if (i < 5)
{
   cout << "liczba mniejsza od 5" ;
   cout << endl ;
   i = 5 ;
}  // tu NIE MA srednika
else
{
   cout << "Liczba nie jest mniejsza od 5" ;
   cout << endl ;
}
```

3.4. Instrukcja while

while (*wyrażenie*) instrukcja ;

Działanie:

Wykonuj instrukcję tak długo jak długo wartość wyrażenia jest niezerowa. Warunek sprawdzany jest przed wykonaniem instrukcji.

Przykład: ❺

```
#include <iostream>
using namespace std;
int main()
{
int ile ;

    cout << "podaj ilosc znakow: " ;
    cin >> ile ;
    // narysuj znaki x
    while (ile)
    {
      cout << "x" ;
      ile = ile - 1 ;
    }
    cout << endl ;
    return 0;
}
```

3.5. Pętla do ... while ...

do instrukcja1 while (*wyrażenie*) ;

Działanie:

Wykonuj instrukcję tak długo jak długo wartość wyrażenia jest niezerowa. Warunek sprawdzany jest po wykonaniu instrukcji.

Przykład: ❻

```
#include <iostream>
using namespace std;
int main()
{
char litera;

    do
    {
      cout << "Podaj litere: " ;
      cin >> litera ;
      cout << "\nNapisales: " << litera << endl ;
    } while (litera != 'k');
    cout << "koniec petli\n" ;
    return 0;
}
```

3.6. Pętla for

for (*instrukcja_inicjalizująca* ; *wyrażenie_warunkowe* ; *instrukcja_kroku*) instrukcja ;

Przykład:

```
for (i = 0 ; i < 10 ; i=i+1)
{
   cout <<  "*" ;
}
```

Działanie:

1. Wykonywana jest instrukcja inicjalizująca pętli.
2. Obliczane jest wyrażenie warunkowe. Jeśli jest równe 0 to praca pętli jest przerywana.
3. Jeżeli wyrażenie warunkowe jest różne od zera, to wykonywane są instrukcje będące treścią pętli.
4. Po wykonaniu treści pętli wykonywana jest instrukcja kroku, po czym powtarzana jest akcja 2.

Przypadki szczególne:

- Instrukcja inicjalizująca nie musi być jedną instrukcją. Może być ich kilka, wtedy oddzielone są przecinkami. Podobnie jest w przypadku instrukcji kroku
- Elementy *instrukcja_inicjalizująca* , *wyrażenie_warunkowe* oraz *instrukcja_kroku* nie muszą wystąpić. Dowolny z nich można opuścić, zachowując średnik oddzielający go od sąsiada. Opuszczenie wyrażenia warunkowego traktowane jest tak, jakby stało tam wyrażenie zawsze prawdziwe.

Przykład ❼:

```
#include <iostream>
using namespace std;
int main()
{
int i, ile ;

   cout << "Podaj liczbe: " ;
   cin >> ile ;
   for (i=1; i<=ile; i=i+1)
   {
     cout << i << ". -*-" ;
     cout << endl ;
   }
   cout << "koniec petli\n" ;
   return 0;
}
```

3.7. Instrukcja switch

switch (*wyrażenie*)
{
case *wart1*:
        intrukcja1 ;
case *wart2*:
        intrukcja2 ;
default:
        instrukcja3 ;
}

Przykład ❽:

```
#include <iostream>
using namespace std;
// rozpoznawanie cyfr
int main()
{
```

```
int liczba;

    cout << "Podaj liczbe: ";
    cin >> liczba;
    switch (liczba)
    {
    case 1:
      cout << "jeden" << endl;
      break;
    case 2:
      cout << "dwa" << endl;
      // tu nie ma break
    case 3:
      cout << "trzy" << endl;
      break;
    default:
      cout << "nie znam" << endl;
      break;
    }
    return 0;
}
```

3.8. Instrukcja break

Instrukcja **break** przerywa działanie pętli:

- for
- while
- do ... while

Przykład:

```
int i=7;
while (1)
{
    i=i-1;
    if (i<5) break;
}
```

3.9. Instrukcja goto

goto *etykieta* ;

Skok do miejsca w programie opatrzonego etykietą.

Przykład:

```
        cout << "tekst 1" ; //zostanie wyswietlony
        cout << "tekst 2" ; //zostanie wyswietlony
        goto etykieta1;
        cout << "tekst 3" ; //nie zostanie wyswietlony
etykieta1:
        cout << "tekst 4" ; //zostanie wyswietlony
```

3.10. Instrukcja continue

Instrukcja **continue** działa wewnątrz pętli:

- for
- while
- do ... while

i powoduje zaniechanie wykonywania instrukcji wewnątrz pętli ale nie przerywa działania pętli.

Przykład:
```
int i;
for (i=0; i<10; i=i+1)
{
   cout << i;
   if (i>5) continue;
   cout << endl;
}
cout << endl;
```

Inaczej można by zapisać:

```
int i;
for (i=0; i<10; i=i+1)
{
   cout << i;
   if (i>5) goto koniec;
   cout << endl;
   koniec:
}
cout << endl;
```

3.11. Klamry w instrukcjach sterujących

Przykład 1:
```
while (i < 4) {
   i = i + 1 ;
}
```
Przykład 2 (Visual Studio 6.0):
```
while (i < 4)
{
     i = i + 1 ;
}
```
Przykład 3:
```
while (i < 4)
     {
     i = i + 1 ;
     }
```

4. Typy

4.1. Deklaracje typów

Deklaracja – informuje kompilator, że dana nazwa reprezentuje obiekt jakiegoś typu, ale nie rezerwuje dla niego miejsca w pamięci.
Definicja - dodatkowo rezerwuje miejsce w pamięci.

Przykład:

```
int a ;  //deklaracja i definicja obiektu typu "int" o nazwie "a"
extern int b;    //deklaracja obiektu typu "int" o nazwie "b";
                 //definicja wystąpi w innym miejscu
```

4.2. Systematyka typów języka C++

- typy fundamentalne
- typy pochodne

- typy wbudowane
- typy zdefiniowane przez użytkownika

4.3. Typy fundamentalne

4.3.1. Typy reprezentujące liczby całkowite

- `short int` inaczej: `short`
- `int`
- `long int` inaczej: `long`

4.3.2. Typ reprezentujący znaki alfanumeryczne

- `char`

4.3.3. Modyfikatory powyższych typów:

- `signed` liczba ze znakiem
- `unsigned` liczba bez znaku (liczba dodatnia)

Przez domniemanie przyjmuje się, iż brak modyfikatora oznacza typ z modyfikatorem *signed*.

4.3.4. Typy reprezentujące liczby zmiennoprzecinkowe

- `float`
- `double`
- `long double`

Przykłady:

```
int a ;
short b ;
short int c ;
long double d ;
unsigned int c ;
```

4.4. Definiowanie obiektów "w biegu"

```
#include <iostream>
using namespace std;
int main()
{
int a ;

   a=1;
   int b; // deklaracja/definicja obiektu
```

```
    b=a+1
    cout << b;
    return 0;
}
```

4.5. Stałe dosłowne

4.5.1. Stałe będące liczbami całkowitymi:

np.: `17 -33 0 1000`

W układzie ósemkowym (zapis rozpoczyna się od cyfry 0):

np.: `010 014 061`

W układzie szesnastkowym (zapis rozpoczyna się od 0x):

np.: `0x10 0xa1 0xff`

Stałe całkowite traktowane są jak typ **int**, chyba że reprezentują liczby, które nie zmieściły by się w typie **int** – wtedy stała jest typu **long**.

Wymuszenie typu stałej:

`200L` - stała typu **long**
`277u` - stała z modyfikatorem **unsigned**
`50uL` - stała **unsigned long**

4.5.2. Stałe reprezentujące liczby zmiennoprzecinkowe

`12.3 -10.1 10.4e-3`

4.5.3. Stałe znakowe

`'a' '7'`

Znaki specjalne:

`'\n'` - nowa linia
`'\t'` - tabulator
`'\\'` - backslash
`'\''` - apostrof
`'\"'` - cudzysłów
`'\0'` - znak o kodzie 0 (NULL)
`'\?'` - znak zapytania

4.5.4. Stałe tekstowe (łańcuchy tekstowe, stringi)

`"To jest łańcuch tekstowy"`

4.6. Typy pochodne

Operatory umożliwiające tworzenie obiektów typów pochodnych:

`[]` – tablica obiektów danego typu
`*` – wskaźnik do obiektu danego typu
`()` – funkcja zwracająca wartość danego typu
`&` – referencja obiektu danego typu

Przykłady:

```
int t[10] ;      // tablica 10 elementów typu int
float *p ;       // wskaźnik do obiektu typu float
char func() ;    //funkcja zwracająca obiekt typu char
```

5.6.1. Typ void

```
void *p ;
```
- oznacza, że p jest wskaźnikiem do obiektu nieznanego typu

```
void funkcja();
```
- funkcja nie zwraca wartości

4.7. Zakres ważności nazwy obiektu, a czas życia obiektu

Czas życia obiektu – okres od momentu zdefiniowania obiektu (przydzielenia miejsca w pamięci) do momentu, gdy przestaje on istnieć (zwolnienia miejsca w pamięci).
Zakres ważności nazwy obiektu – część programu, w której nazwa znana jest kompilatorowi.

4.8. Zasłanianie nazw

Przykład ❾:

```
#include <iostream>
using namespace std;
int k=33;  // zmienna globalna
int main()
{
   cout << "main (przed blokiem): k=" << k << endl;
   { //poczatek bloku: zakres lokalny
     int k=1;
     cout << "blok: k=" << k << endl;
     cout << "dostep do zmiennej globalnej k=" <<
                 ::k << endl;
   } //koniec bloku
   cout << "main (za blokiem): k=" << k << endl;
   return 0;
}
```

4.9. Modyfikator const

Służy do tworzenia obiektów stałych, których wartości nie da się zmienić.

```
const float pi=3.1415927;
```

Wartości tak zainicjalizowanego obiektu nie można zmienić.

Inicjalizacja – nadanie wartości obiektowi w momencie jego utworzenia.
Przypisanie – wstawienie do obiektu wartości w jakimkolwiek późniejszym momencie.

Obiekty **const** można inicjalizować, ale nie można do nich nic przypisać.

4.10. Instrukcja typedef

Pozwala na nadanie dodatkowej nazwy już istniejącemu typowi.

```
typedef int cena ;
typedef char * napis ;
cena x;         //co odpowiada: int x;
napis komunikat;//co odpowiada: char * komunikat;
```

4.11. Typy wyliczeniowe enum

enum *nazwa_typu* {*lista_wyliczeniowa*} ;

Przykład:

```
enum liczby { zero, jeden, dwa, trzy };
enum liczby2
{
```

```
      jedenascie=11,
      trzynascie=13,
      dwadziescia=20
   };
   liczby a;        //definicja zmiennej
   liczby2 b;       //definicja zmiennej

      a=zero;
      b=dwadziescia;
```

---

5. Operatory

5.1. Operatory arytmetyczne

`+`	dodawanie
`-`	odejmowanie
`*`	mnożenie
`/`	dzielenie

Przykład:

```
a = b + c ;
a = b - c ;
a = b * c ;
a = b / c ;
a = (c + d + 3.1) / f ;
```

5.1.1. Operator modulo

Oblicza resztę z dzielenia

```
a = b % c ;
```

5.1.2. Jednoargumentowe operatory + i –

Przykład:

```
+12.7          -x          -(a*b)
```

5.1.3. Operatory inkrementacji i dekrementacji

`i = i + 1 ;`	można zastąpić	`i++ ;`
`k = k - 1;`	można zastąpić	`k--;`

Formy operatorów:

- przedrostkowa (prefix): `++a` lub `--b` . Najpierw zmieniana jest wartość zmiennej, a następnie zmieniona wartośc staje się wartością wyrażenia.
- końcówkowa (postfix): `a++` lub `b--` . Wartość zmiennej staje się wartością wyrażenia, a następnie zmieniana jest wartość zmiennej.

Przykład:

```
int a=0;
int b=0;
cout << ++a << endl;  // pojawi się 1
cout << b++ << endl;  // pojawi się 0
cout << a << endl;    // pojawi się 1
cout << b << endl;    // pojawi się 1
```

5.1.4. Operator przypisania =

```
m = 27.9
```

Do obiektu stającego po lewej stronie operatora = wstawiona zostaje wartość wyrażenia stojącego po prawej.

5.2. Operatory logiczne

5.2.1. Operatory relacji

`<`	mniejszy niż ...
`<=`	mniejszy lub równy
`>`	większy niż ...
`>=`	większy lub równy
`!=`	różny
`==`	równy

Przykład ❶❷:

```
#include <iostream>
using namespace std;
int main()
{
int a=10;
int b=5;

   cout << "a=" << a << " b=" << b << endl;
   if (a=b)     // tu powinno byc a==b
     cout << "a=b" << endl;
   else
     cout << "a!=b" << endl;
   return 0;
}
```

Powyższy program wyświetli informacje, że a=b, gdyż w wyrażeniu warunkowym instrukcji if znajduje się instrukcja przypisania a nie operator relacji. Instrukcja przypisania jest wyrażeniem, a więc ma wartość (wartość wyrażenia po prawej stronie znaku =). W tym przypadku jest to wartość 5, która jest różna od zera, a więc zostanie zinterpretowana jako "prawda".

5.2.2. Operatory sumy i iloczynu logicznego

`||`	suma logiczna – operacja logiczna "lub" (alternatywa)
`&&`	iloczyn logiczny – operacja logiczna "i" (koniunkcja)

Przykład:

```
int k=2;
if ((k==10) || (k==2))
   cout << "k równe 2 lub 10 \n" ;
if ((k>=0) && (k<=10))
   cout << "k nalezy do przedzialu [0,10] \n" ;
```

5.2.3. Operator negacji logicznej

! wyrażenie

Przykład:

```
int i = 0 ;
if (!i)
   cout << "tekst\n" ;
```

5.3. Operatory bitowe

zmienna << *ile_miejsc*	przesunięcie w lewo
zmienna >> *ile_miejsc*	przesunięcie w prawo
zmienna1 & *zmienna2*	bitowy iloczyn logiczny (AND)
zmienna1 | *zmienna2*	bitowa suma logiczna (OR)
zmienna1 ^ *zmienna2*	bitowa różnica symetryczna (XOR)
~ *zmienna*	bitowa negacja

5.4. Pozostałe operatory przypisania

operator	zamiennik
`i = i + 2`	`i += 2`
`i = i - 2`	`i -= 2`
`i = i * 2`	`i *= 2`
`i = i / 2`	`i /= 2`
`i = i % 2`	`i %= 2`
`i = i >> 2`	`i >>= 2`
`i = i << 2`	`i <<= 2`
`i = i & 2`	`i &= 2`
`i = i | 2`	`i |= 2`
`i = i ^ 2`	`i ^= 2`

5.5. Wyrażenie warunkowe

(*warunek*) ? *wartość1* : *wartość2*

Jeżeli *warunek* ma wartość niezerową to wartością wyrażenia stanie się *wartość1* – w przeciwnym wypadku *wartość2*.

Przykład:

```
int a;
a=5;
c = (a<10) ? 0 : 100 ;      // zmiennej c zostanie przypisane 0
```

5.6. Operator sizeof

Zwraca rozmiar obiektu lub typu.

```
sizeof(nazwa_typu)
```

lub

```
sizeof(nazwa_obiektu)
```

Zwrócona liczba jest typu `size_t` (inaczej: `unsigned int`).

Gdy argumentem operatora jest nazwa tablicy statycznej zwraca rozmiar tablicy w bajtach. Nie jest w stanie określić rozmiaru tablicy alokowanej dynamicznie oraz tablicy będącej parametrem formalnym funkcji.

5.7. Operator rzutowania

```
(nazwa_typu)obiekt
```

lub

```
nazwa_typu(obiekt)
```

Umożliwia przekształcenie typu obiektu.

Przykład:

```
int a;
float b=10.1;
a=(int)b;
cout << a << endl;    //zostanie wyswietlona wartosc 10
```

5.8. Operator przecinek

Umożliwia grupowanie wyrażeń.

```
(2+4, a*4, 3<6, 77+2)
```

Wyrażenia składowe obliczane są od lewej do prawej. Wartością całego wyrażenia staje się wartość wyrażenia znajdującego się najdalej z prawej strony.

5.9. Programy przykładowe

Przykład ❶❶:
Temat: Program dodający dwie liczby rzeczywiste.

```
#include <iostream>
using namespace std;
int main()
{
float a, b, wynik;

   cout << "Podaj pierwsza liczbe: ";
   cin >> a;
   cout << "Podaj druga liczbe: ";
   cin >> b;
   wynik=a+b;
   cout << "Suma: " << wynik << endl;
   return 0;
}
```

Przykład ❶❷:
Temat: Drukowanie alfabetu.

```
#include <iostream>
int main()
{
char znak;

   for (znak='a'; znak<='z'; znak++)
     cout << znak ;
   cout << endl ;
   return 0;
}
```

Przykład ❶❸:
Temat: Obliczanie *n!* .

```
#include <iostream>
using namespace std;
int main()
{
unsigned long silnia, liczba, licz;

   cout << "Podaj liczbe: ";
   cin >> liczba;
   silnia=1;
   licz=2;
   if (liczba)
     while (licz<=liczba) //for ( ; licz<=liczba; )
     {
          silnia *= licz;
```

```
            licz ++;
        }
      cout << liczba << "! =" << silnia << endl;
      return 0;
   }
```

Przykład ❶❹:
Temat: Wyjście z pętli gdy wprowadzona liczba jest równa zero.

```
#include <iostream>
using namespace std;
int main()
{
int liczba;

   do
   {
     cout << "Podaj liczbe: ";
     cin >> liczba;
     if (!liczba) cout << "zero - koniec petli\n";
     else cout << "To nie jest zero\n";
   } while (liczba);
   return 0;
}
```

Przykład ❶❺:
Temat: Program pobiera dwie liczby a następnie kod operacji: 0 – koniec; 1 – suma; 2 – różnica; 3 – iloczyn; 4 – iloraz. Wyświetla wynik obliczeń.

```
#include <iostream>
int main()
{
float a, b;
int kod;

   do
   {
     cout << "Wprowadz a= ";
     cin >> a;
     cout << "Wprowadz b= ";
     cin >> b;
     cout << "0:koniec; 1:(+); 2:(-); 3:(*); 4:(/)." <<
                   "Podaj kod: ";
     cin >> kod;
     switch (kod)
     {
     case 0:
          cout << "Koniec pracy\n";
          break;
     case 1:
          cout << a << "+" << b << "=" << a+b << endl;
          break;
     case 2:
          cout << a << "-" << b << "=" << a-b << endl;
          break;
```

```
          case 3:
               cout << a << "*" << b << "=" << a*b << endl;
               break;
          case 4:
               cout << a << "/" << b << "=" << a/b << endl;
               break;
          default:
               cout << "Nieznany kod operacji\n";
               break;
          }
     } while (kod);  //while (kod!=0);
return 0;
}
```

6. Funkcje

Przykład ❶❻:

```
#include <iostream>
using namespace std;
char literki(int ile);          //deklaracja funkcji
//--------------------------------------------
int main()
{
int m;
char znak;
   cout << "Podaj ilosc literek: ";
   cin >> m;
   znak=literki(m);
   cout << "Drukowano litere " << znak << endl;
   return 0;
}
//--------------------------------------------
// Definicja funkcji
char literki(int ile)
{
int i;
   for (i=0; i<ile; i++)
     cout << "f";
   cout << endl;
   return 'f';
}
```

6.1. Zwracanie rezultatu przez funkcję

Przykład ❶❼: Temat: Program obliczający potęgi liczb całkowitych z podanego przedziału.

```
#include <iostream>
using namespace std;
long potega(long liczba, int stopien);
void koniec();
//-----------------------
int main()
{
long pocz, kon, i;
   cout << "Podaj poczatek przedzialu: ";
   cin >> pocz;
```

```
    cout << "Podaj koniec przedzialu: ";
    cin >> kon;
    for (i=pocz; i<=kon; i++)
    {
      cout << i << "^2=" << potega(i,2) << "\t";
      cout << i << "^3=" << potega(i,3) << endl;
    }
    koniec();
    return 0;
}
//---------------------------
long potega(long liczba, int stopien)
{
long wynik=liczba;
int i;
    for (i=1; i<stopien; i++)
      wynik *= liczba;
    return wynik;
}//----------------
void koniec()
{
    cout << "\nKONIEC PRACY\n";
}
```

6.2. Przesyłanie argumentów

Rodzaje przesyłania argumentów do funkcji:

- przez wartość,
- przez referencję,
- przez wskaźnik.

Przykład ❶❽:
Temat: przesyłanie argumentów przez wartość i przez referencję.

```
#include <iostream>
using namespace std;
//----------------------------
void zeruj(int a, int &b)
{
    cout << "-------------\n";
    cout << "Funkcja\n";
    cout << "a=" << a << "\tb=" << b << endl;
    a=0; b=0;
    cout << "Po wyzerowaniu:\n";
    cout << "a=" << a << "\tb=" << b << endl;
    cout << "-------------\n";
    return;
}
//----------------------------
int main()
{
int A=7, B=77;
    cout << "Main\n";
    cout << "A=" << A << "\tB=" << B << endl;
    zeruj(A,B);
    cout << "Main\n";
```

```
    cout << "Po wywolaniu funkcji:\n";
    cout << "A=" << A << "\tB=" << B << endl;
    return 0;
}
```

6.3. Argumenty domniemane

Przykład:
Deklaracje funkcji:

```
void funkcja1(int arg1, int arg2=0);
void funkcja2(float arg1, int arg2=1, int arg3=5);
```

Wywołanie funkcji:

```
funkcja1(10);          ➔   funkcja1(10,0);
funkcja1(-10,5);
funkcja2(-2.1,-5);     ➔   funkcja2(-2.1,-5,5);
funkcja2(-1, ,3);      ➔   tak jest ŹLE
```

Określenie argumentów domniemanych musi wystąpić w deklaracji funkcji lub w definicji, która jest jednocześnie deklaracją.

6.4. Funkcje inline

Przykład:

```
inline int dodaj(int a, int b)
{
    return a+b;
}
```

Kompilator utworzy kod, w którym w miejscu wywołania funkcji zostanie wstawiony kod definiujący funkcję, a nie skok do miejsca definicji funkcji.

Definicja funkcji *inline* musi znajdować się przed jej wywołaniem, gdyż w momencie wywołania funkcja musi być znana kompilatorowi – nie wystarczy deklaracja funkcji.

6.5. Zakresy ważności nazw deklarowanych wewnątrz funkcji

- Zakres ważności nazw deklarowanych w obrębie funkcji ogranicza się tylko do bloku tej funkcji. Nie można spoza funkcji za pomocą danej nazwy próbować dotrzeć do zmiennej będącej w obrębie funkcji
- Nie można wykonać instrukcji *goto* spoza funkcji do etykiety zdefiniowanej wewnątrz funkcji.

6.6. Wybór zakresu ważności nazwy i czasu życia obiektu

6.6.1. Obiekty globalne

Obiekty zadeklarowane na zewnątrz wszystkich funkcji.
Zakres ważności: plik programu.
Czas życia: czas działania programu.
Wartość początkowa: 0

6.6.2. Obiekty automatyczne

Obiekty zadeklarowane wewnątrz bloku programu (np. wewnątrz funkcji, bloku instrukcji).
Zakres ważności: blok programu.

Czas życia: od momentu wystąpienia definicji do końca bloku.
Wartość początkowa: przypadkowa.

6.6.2. Obiekty lokalne statyczne

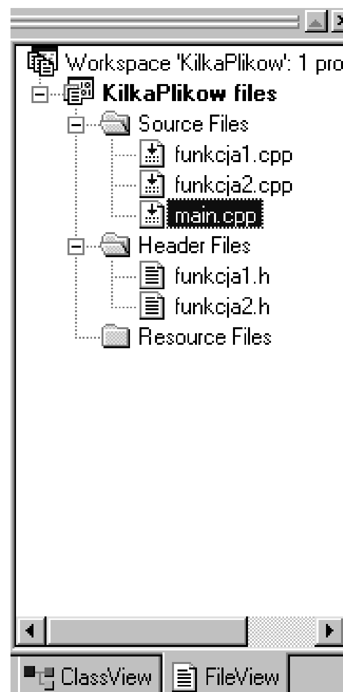
Obiekty zadeklarowane wewnątrz bloku programu (np. wewnątrz funkcji, bloku instrukcji) i poprzedzone słowem `static`.
Przykład:

```
static int a;
static float b=2.1;
```

Zakres ważności: blok programu.
Czas życia: czas działania programu.
Wartość początkowa: 0.

6.7. Obiekty w programie składającym się z kilku plików

Przykład ❶❾:



Plik *main.cpp*:

```
#include <iostream>
using namespace std;
#include "funkcja1.h"
#include "funkcja2.h"
int a;     //definicja zmiennej globalnej
int main()
{
int x, y;
   cout << "Podaj zmienna globalna a=";
   cin >> a;
   cout << "Podaj x=";
   cin >> x;
   cout << "Podaj y=";
   cin >> y;
   Wypisz_a();
   cout << "x+y=" << dodaj(x,y) << endl;
   cout << "x-y=" << odejmij(x,y) << endl;
return 0;
}
```

Plik *funkcja1.cpp*:

```
#include <iostream>
using namespace std;
extern int a;   //deklaracja zmiennej globalnej
//-------------------------
int dodaj(int a, int b)
{
   return a+b;
}
//-------------------------
```

```
void Wypisz_a()
{
   cout << "\nZmienna globalna a=" << a << endl;
}
```

Plik *funkcja2.cpp:*

```
int odejmij(int a, int b)
{
   return a-b;
}
```

Plik *funkcja1.h*:

```
int dodaj(int a, int b);
void Wypisz_a();
```

Plik *funkcja2.h*:

```
int odejmij(int a, int b);
```

6.7.1. Nazwy statyczne globalne

Deklaracje globalne poprzedzone słowem `static`.
Nazwa ta nie będzie znana w innych plikach.

7. Preprocesor

7.1. Dyrektywa #define

#define *wyraz* ciąg znaków zastępujących go

Kompilator **zamieni** wszelkie wystąpienia *wyrazu* ciągiem znaków zastępujących go.

Przykład 1:

```
#define DWA 2
i=DWA + 2;
```

Przykład 2:

```
#define DO_START      do {
#define DO_STOP       }while (!stop);
DO_START
   a=a+1;
   cin >> stop;
DO_STOP
```

7.2. Makrodefinicje

#define *wyraz(parametry)* ciąg znaków zastępujących go

Kompilator **zamieni** wszelkie wystąpienia *wyrazu* ciągiem znaków zastępujących go uwzględniając parametry. Stąd w makrodefinicjach najczęściej występuje nadmiar nawiasów, aby ustrzec się przed potencjalnymi błędami.

Przykład:

```
#define KWADRAT(a)    ((a) * (a))
wynik=KWADRAT(liczba);
```

zostanie zamienione na:

```
wynik=((liczba) * (liczba)) ;
```

7.3. Dyrektywy kompilacji warunkowej

```
#if warunek
        linie kompilowane warunkowo
#endif
```

lub

```
#if warunek
        linie kompilowane warunkowo
#else
        linie kompilowane warunkowo
#endif
```

lub

```
#ifdef nazwa
        linie kompilowane warunkowo
#else
        linie kompilowane warunkowo
#endif
```

lub

```
#ifndef nazwa
        linie kompilowane warunkowo
#else
        linie kompilowane warunkowo
#endif
```

Przykład 1:

```
#define WARIANT 1
#if (WARIANT==1)
   // linie programu
   // .............
#else
   // linie programu
   // .............
#endif  //WARIANT==1
```

7.4. Wstawianie innych plików

```
#include <nazwa_pliku_1>
#include "nazwa_pliku_2"
```

Przykład:
Zabezpieczenie przed wielokrotnym dołączeniem pliku:

```
#ifndef PLIK1_H
#define PLIK1_H
   // zwykla tresc pliku
   // ..................
#endif //PLIK1_H
```

7.5. Dyrektywa #undef

Przykład

```
#define ABC          // zaczyna obowiazywac nazwa ABC
// ...................
// linie programu
```

```
// .....................
#undef ABC      // przestaje obowiazywac nazwa ABC
// .....................
```

8. Tablice

Definicja tablicy:

typ *nazwa*[rozmiar];

Rozmiar tablicy musi być stałą znaną w trakcie kompilacji.

Przykłady:

```
char zdanie[80];// tablica o nazwie zdanie zawierajaca
                // 80 elementów typu char
float numer[9]; // tablica 9 elementow typu float
int *wskaz[20]; // tablica 20 element bedacych wskaznikami
                // obiektow typu int
```

Tablice można tworzyć z:

- typów fundamentalnych (za wyjątkiem *void*),
- typów wyliczeniowych (*enum*),
- wskaźników,
- innych tablic,
- z obiektów typu zdefiniowanego przez użytkownika,
- ze wskaźników do pokazywania na składniki klasy.

8.1. Elementy tablicy

Tablica:

```
int t[4];
```

składa się z następujących elementów:

```
t[0] t[1] t[2] t[3]
```

Numeracja elementów tablicy zaczyna się od zera.

Dla zdefiniowanej powyżej tablicy możliwe są m.in. następujące instrukcje przypisania:

```
t[0]=1;
t[1]=23;
t[2]=-5;
t[3]=10;
```

Poniższe przypisanie **RÓWNIEŻ ZOSTANIE WYKONANE**:

```
t[4]=1;
```

Kompilator C++ nie sprawdza zakresu indeksów tablicy

Sposób obliczenia rozmiaru tablicy (nie dotyczy tablicy dynamicznej i parametru formalnego funkcji):

```
size_t rozmiar = sizeof(tab)/sizeof(tab[0]);
```

Przykład ❷⓿:

```
#include <iostream>
using namespace std;
int main()
{
int t[4];
```

```
    for (int i=0; i<4; i++)
      t[i]=2*i;
    for (i=0; i<4; i++)
      cout << "t[" << i << "]=" << t[i] << endl;
    return 0;
}
```

8.2. Inicjalizacja tablic

Przykłady:

```
int tab1[4] = { 17, 5, 4, 200 };
int tab2[4] = { 3,-2 };     // pozostałe elementy zostaną
                            // zainicjalizowane wartoscia 0
int tab3[] = {-3, 1, 6, 8 };     // zostanie zdefiniowana
                                 // tablica "int tab3[4]"
```

8.3. Przekazywanie tablic do funkcji

Nazwa tablicy jest równocześnie adresem jej zerowego elementu.

Tablice przesyła się podając funkcji tylko adres początku tablicy.

Przykład ❷❶:

```
#include <iostream>
using namespace std;
void dodaj2(int tablica[], int ile);
//----------------------------
int main()
{
const int rozmiar=4;
int tab[rozmiar];     // można wykorzystac zmienna rozmiar
                      // gdyz jest to zmienna z modyfikatorem
                      // const
int i;
   for (i=0; i<rozmiar; i++)
     tab[i]=i;
   dodaj2(tab,rozmiar);
   for (i=0; i<rozmiar; i++)
     cout << "tab[" << i << "]=" << tab[i] << endl;
   return 0;
}
//-------------------------------
void dodaj2(int tablica[], int ile)
{
int i;
   for (i=0; i<ile; i++)
     tablica[i] += 2;
}
```

Przykład ❷❷:

```
#include <iostream>
using namespace std;
#define ROZMIAR 4
void dodaj2(int tablica[ROZMIAR]);
//----------------------------
int main()
{
int tab[ROZMIAR];
```

```
int i;
   for (i=0; i<ROZMIAR; i++)
     tab[i]=i;
   dodaj2(tab);
   for (i=0; i<ROZMIAR; i++)
     cout << "tab[" << i << "]=" << tab[i] << endl;
   return 0;
}
//------------------------------
void dodaj2(int tablica[ROZMIAR])
{  // ROZMIAR nie jest konieczny, gdyz funkcja nie korzysta z
tej
   // informacji
int i;
   for (i=0; i<ROZMIAR; i++)
     tablica[i] += 2;
}
```

W funkcji do określenia rozmiaru tablicy nie można wykorzystać operatora `sizeof`, gdyż na podstawie parametru formalnego nie da się określić rozmiaru elementu.

8.4. Tablice znakowe

Łańcuch znaków (string) – ciąg znaków alfanumerycznych zakończony znakiem o kodzie 0 (\0 – NULL).

Przykłady tablic znakowych:

```
char zdanie[80];// tablica do przechowywania 80 elementów
                // będących znakami.
char sTab[10] = { "lancuch" };
```

sTab:

l	a	n	c	u	c	h	NULL	0	0

Inne sposoby inicjalizacji:

```
char sTab[10] = { 'k', 'o', 't' };
```

k	o	t	0	0	0	0	0	0	0

Ponieważ NULL ma kod 0, więc łańcuch zostanie poprawnie zakończony.

```
char sTab[] = { 'k', 'o', 't' };
```

k	o	t

Kompilator określi rozmiar tablicy na podstawie liczby elementów na liście inicjalizacyjnej. Nie ma tam znaku NULL, a więc nie jest to tablica przechowująca łańcuch znakowy.

```
char sTab[] = { "kot" };
```

k	o	t	NULL

Na liście inicjalizacyjnej znajduje się łańcuch znakowy, więc kompilator automatycznie doda znak NULL.

W powyższy sposób można jedynie inicjalizować tablice. Nie można wykonywać operacji przypisania.

Przykład ❷❸:
Temat: Funkcja kopiująca łańcuchy znakowe.

```
#include <iostream>
using namespace std;
void kopiuj(char cel[], char zrodlo[]);
void wypisz(char tab[]);
//---------------------------
int main()
{
char sTab1[] = { "kot"};
char sTab2[10];
   cout << "sTab1=";
   wypisz(sTab1);
   kopiuj(sTab2,sTab1);
   cout << "sTab2=";
   wypisz(sTab2);
return 0;
}
//-----------------------------
void kopiuj(char cel[], char zrodlo[])
{
int i;
   for (i=0; ; i++)
   {
     cel[i]=zrodlo[i];
     if (zrodlo[i]==NULL) break;
   }
}
//-----------------------------
void wypisz(char tab[])
{
int i;
   for (i=0; ; i++)
     if (tab[i]==NULL) break;
     else cout << tab[i];
   cout << endl;
}
```

Przykład ❷❹:
Temat: Program wykorzystujący funkcję biblioteczną kopiująca łańcuchy znakowe.

```
#include <iostream>
#include <string>
using namespace std;
void wypisz(char tab[]);
//---------------------------
int main()
{
char sTab1[] = { "kot"};
char sTab2[10];
   cout << "sTab1=";
   wypisz(sTab1);
   strcpy(sTab2,sTab1);
   cout << "sTab2=";
   wypisz(sTab2);
   return 0;
```

```
}
//----------------------------
void wypisz(char tab[i]) // taka jak w poprzednim programie
```

8.5. Tablice wielowymiarowe

Są to tablice, których elementami są inne tablice.

```
int Tab[4][2];  // tablica czterech elementów, z których
                // każdy jest dwuelementową tablicą
                // obiektów typu int
```

Uwaga: zapis `Tab[i,j]` zostanie zinterpretowany jako `Tab[j]` (występuje operator "przecinek").

8.5.1. Inicjalizacja tablic wielowymiarowych:

int Tab1[3][2] = { 11, 12, 21, 22, 31, 32 };
int Tab2[3][2] = { {11, 12}, {21, 22}, {31, 32} };

Element Tab1[1,1] jest przesunięty w stosunku do początku tablicy o `(1*2)+1` elementów, czyli `numer_wiersza*ilość_kolumn+numer_kolumny` – aby obliczyć jego położenie musi być znana liczba kolumn tablicy (drugi wymiar) – nie jest konieczna znajomość ilości wierszy (pierwszy wymiar).

	0	1
0	11	12
1	21	22
2	31	32

8.5.2. Przesyłanie tablic wielowymiarowych do funkcji

Aby funkcja mogła obliczyć, gdzie w pamięci znajduje się określony element tablicy – musi znać liczbę kolumn tablicy.

```
void funkcja1(float tab[][4]);// funkcja musi znać liczbę
                              //  kolumn
void funkcja2(float tab[][3][5]);  //   aby obliczyć położenie
                                   //     elementu
// void funkcja3( float tab[][][]) tak jest źle
```

9. Wskaźniki

Wskaźnik to obiekt do przechowywania informacji o adresie i typie innego obiektu.

9.1. Definiowanie wskaźników

```
int * w ;        // w jest wskaźnikiem do obiektów typu int
char * pLancuch;// pLancuch jest wskaznikiem do obiektow typu char
float * pWsk;    // pWsk jest wskaznikiem do obiektow typu float
```

Zastosowanie wskaźników:

- ulepszenie pracy z tablicami,
- funkcje mogące zmieniać wartość przysyłanych do nich argumentów,

- dostęp do specjalnych komórek pamięci,
- rezerwacja obszarów pamięci.

9.2. Praca ze wskaźnikiem

Przykład ❷❺:

```
#include <iostream>
using namespace std;
//----------------------------
int main()
{
int nLiczba=5;
int nLiczba2=10;
int *pInt;
int *pInt2;
int *pInt3;
float Liczba3=1.2;
float *pFloat;
void *pWsk;

   cout << "nLiczba=" << nLiczba << endl; // pojawi sie 5
   pInt = &nLiczba;  // ustawienie wskaznika
   cout << "pInt=" << pInt << endl; // pojawi sie adres
   cout << "(*pInt)=" << *pInt << endl; // pojawi sie 5
   cout << "Przypisanie *pInt=6\n" ;
   *pInt = 6 ;
   cout << "(*pInt)=" << *pInt << endl; // pojawi sie 6
   cout << "nLiczba=" << nLiczba << endl; // pojawi sie 6
   cout << "zmiana obiektu wskazywanego\n" ;
   pInt = &nLiczba2;
   cout << "nLiczba2=" << nLiczba2 << endl; // pojawi sie 10
   cout << "(*pInt)=" << *pInt << endl; // pojawi sie 10
   // --------------------------------
   pFloat = &Liczba3;
   cout << "(*pFloat)=" << *pFloat << endl; // pojawi sie 1.2
   // --------------------------------
   // pFloat=pInt; // <<< - zostanie zasygnalizowany blad
   pWsk = pInt;
   cout << "pWsk=" << pWsk << endl; // pojawi sie adres
   // cout << *pWsk; // <<< - blad
   pInt2 = (int*)pWsk;  // musi byc rzutowanie
   cout << "(*pInt2)=" << *pInt2 << endl; // pojawi sie 10

   pFloat = (float*)pInt; // musi byc rzutowanie
   cout << "(*pFloat)=" << *pFloat << endl; // pojawi sie
           // przypadkowa liczba mimo prawidlowego adresu
   pInt3 = (int*)pFloat; // musi byc rzutowanie
   cout << "(*pInt3)=" << *pInt3 << endl; // pojawi sie 10
return 0;
}
```

Wskaźnik do obiektu każdego niestałego typu można przypisać wskaźnikowi typu void.

9.3. Zastosowanie wskaźników wobec tablic

Przykład:

```
int *wsk1, *wsk2;
int tab1[3]={11, 12, 13 };
int tab2[3]={21, 22, 23 };
wsk1 = &tab1[0];
wsk2 = tab2;  // nazwa tablicy jest adresem jej pierwszego
              // elementu
wsk1 = &tab1[2]; // wskaznik pokazuje na ostatni element
                 // tablicy
wsk2 = wsk2 + 1; // wskaznik pokazuje na drugi element
                 // tablicy
```

Dodanie do wskaźnika liczby całkowitej powoduje, że pokazuje on o tyle dalszy element tablicy niezależnie od tego jakiego typu są to elementy (niezależnie od ilości pamięci przeznaczonej na każdy element).

Przykład ❷❻:

```
#include <iostream>
using namespace std;
int main()
{
int *wsk1, *wsk2;
int tab[10];
int i;

    wsk1=tab;
    for (i=0; i<10; i++)
      *(wsk1+i)=i*10;
    cout << "Elementy tablicy:\n";
    for (i=0; i<10; i++)
      cout << i << ": " << *(wsk1+i) << endl;
    cout << endl;
    wsk2 = &tab[3];
    cout << "3: " << *wsk2 << endl;
    wsk2 += 2;
    cout << "(3+2): " << *wsk2 << endl;
    return 0;
}
```

9.4. Działania na wskaźnikach

(wskaźnik + liczba_całkowita) – przesunięcie wskaźnika o podaną liczbę elementów w przód,
(wskaźnik – liczba_całkowita) – przesunięcie wskaźnika o podaną liczbę elementów w tył,
(wskaźnik – wskaźnik) – gdy wskaźniki pokazują na elementy tej samej tablicy to wynikiem będzie liczba dzielących je elementów (liczba dodatnia lub ujemna).

9.5. Porównywanie wskaźników

operator `==` równość wskaźników oznacza, że pokazują na ten sam obiekt,
operator `!=` przeciwieństwo operatora `==`,

Operatory mające zastosowanie do wskaźników pokazujących na elementy tej samej tablicy:

```
>    <    >=    <=
```

9.6. Zastosowanie wskaźników w argumentach funkcji

Przykład ❷❼:

```
#include <iostream>
using namespace std;
void zmien(int *wsk);
//---------------------
int main()
{
int liczba=1;
   cout << "Przed wywolaniem funkcji: " << liczba << endl;
   zmien(&liczba);
   cout << "Po wywolaniu funkcji: " << liczba << endl;
   return 0;
}
//---------------------
void zmien(int *wsk)
{
   *wsk=100;
}
```

9.6.1. Odbieranie tablicy jako wskaźnika

Przykład ❷❽:

```
#include <iostream>
using namespace std;
void fkc_tab(int tab[], int rozmiar);
void fkc_wsk1(int *wsk, int rozmiar);
void fkc_wsk2(int *wsk, int rozmiar);
//---------------------
int main()
{
int tablica[4] = { 1, 2, 3, 4 };
   fkc_tab(tablica,4);
   fkc_wsk1(tablica,4);
   fkc_wsk2(tablica,4);
   cout << endl;
return 0;
}
//---------------------
void fkc_tab(int tab[], int rozmiar)
{
   cout << "\nPrzeslano jako tablice\n" ;
   for (int i=0; i<rozmiar; i++)
     cout << tab[i] << "\t";
}
//---------------------
void fkc_wsk1(int *wsk, int rozmiar)
{
   cout << "\nPrzeslano jako wskaznik - wariant 1\n" ;
   for (int i=0; i<rozmiar; i++)
     cout << *(wsk+i) << "\t";
}
//---------------------
void fkc_wsk2(int *wsk, int rozmiar)
```

```
{
   cout << "\nPrzeslano jako wskaznik - wariant 2\n" ;
   for (int i=0; i<rozmiar; i++)
     cout << wsk[i] << "\t";
}
```

9.6.2. Argument formalny będący wskaźnikiem do obiektu const

Przykład:

```
void funkcja(const int *wsk, int rozmiar);
```

Wskaźnik *const* pokazuje na obiekty, ale nie pozwala na ich modyfikacje.
Zastosowanie:

1. Zagwarantowanie, że obiekt wysłany do funkcji poprzez wskaźnik nie zostanie zmieniony,
2. Umożliwienie wysłania do funkcji poprzez wskaźnik obiektu stałego (można na niego pokazać tylko wskaźnikiem do stałej).

9.7. Zastosowanie wskaźników przy dostępie do konkretnych komórek pamięci

Przykład:

```
wsk = 0x0065fde8;
```

Wskaźnikowi należy przypisać adres komórki pamięci (sposób adresowania zależny jest od architektury komputera, systemu operacyjnego i kompilatora).

9.8. Rezerwacja obszarów pamięci

9.8.1. Operatory new i delete

Przykład:

```
int *wsk;
wsk = new int;
// ...........
delete wsk;
```

Operator `new` utworzy nowy obiekt typu `int`, który nie będzie miał nazwy, a jego adres zostanie przekazany do wskaźnika `wsk`. Operator `delete` zlikwiduje obiekt pokazywany przez wskaźnik `wsk`.
Dynamiczne tworzenie tablic:

```
float *wsk;
int rozmiar;
cout << "Podaj rozmiar tablicy: "; cin >> rozmiar;
wsk = new float[rozmiar];
//....
delete [] wsk;
```

Wstępna inicjalizacja obiektu:

```
int *wsk;
wsk = new int(12);
```

Alokacja obiektu w konkretnym miejscu pamięci:

```
wsk = 0x0065fde8;
wsk2 = wsk new int;
```

Cechy obiektów utworzonych operatorem `new`:

- obiekty te nie mają nazwy – dostęp do nich odbywa się poprzez wskaźnik,
- początkowe wartości obiektów są przypadkowe, chyba że zostały wstępnie zainicjalizowane,
- czas życia: od chwili utworzenia operatorem `new` do momentu usunięcia operatorem `delete`,
- zakres ważności: jeśli w danym momencie jest przynajmniej jeden wskaźnik pokazujący na obiekt, to jest do niego dostęp.

Przykład ❷❾:

```
#include <iostream>
using namespace std;
int main()
{
int *wsk1, *wsk2;
int i, rozmiar;
   cout << "Podaj rozmiar tablicy: ";
   cin >> rozmiar;
   wsk1 = new int[rozmiar];
   if (wsk1==NULL)
     cout << "Brak pamieci\n";
   else
   {
     for (i=0; i<rozmiar; i++)
          wsk1[i]=i*10;
     cout << "Tablica:\n";
     for (i=0; i<rozmiar; i++)
          cout << i << ": " << wsk1[i] << endl;
     wsk2 = new int;
     *wsk2=10;
     cout << "*wsk2 = " << *wsk2 << endl;
     wsk2=wsk1; // utrata dostepu do obiektu
     delete [] wsk1;
     wsk1=NULL;
     //delete wsk2; // niebezpieczenstwo zalamania programu
   }
return 0;
}
```

Wskaźniki definiowane z modyfikatorem `static` inicjalizowane są wartością `NULL`. Pozostałe wskaźniki mają wartości przypadkowe, a więc wskazują na przypadkowe obszary pamięci. Próba zapisu takiego miejsca może spowodować załamanie programu.

9.9. Sposoby inicjalizowania wskaźników

```
wsk = & obiekt;

wsk = inny_wskaznik;

wsk = tablica;

wsk = funkcja;
```

```
wsk = new int;

wsk = new float[10];

wsk = 0x82a5f2;  // adres

wsk = "lancuch tekstowy";
```

9.10. Tablice wskaźników

Przykład – pięcioelementowa tablica wskaźników do obiektów typu `float`:

```
float *tabwsk[5]; // inaczej: float *(tabwsk[5]);
```

Przykład – tablice wskaźników do stringów:

```
char *tab1[3];
char *tab2[3] = { "jeden", "dwa", "trzy" };
```

W tablicy `tab2` znajdują się wskaźniki do miejsc w pamięci, w których zlokalizowane są poszczególne łańcuchy tekstowe.

Przykład ❸⓿:

```
#include <iostream>
using namespace std;
char* dopisz_spacje(const char *wsk1, char *wsk2);
char* dopisz_spacje2(const char *wsk1, char **wsk2);
void kody(const char *wsk, int rozmiar);
//--------------------
int main()
{
char tab1[] = { "tablica1" };
char tab2[20] = { 't','a','b','l','i','c','a','2' };
char tab3[] = { 't','a','b','l','i','c','a','3','\0' };
char *wsk1 = "wskaznik1";
char *wsk2 = { "wskaznik2" };
char *wsk=NULL;

   cout << "tab1: " << tab1 << endl;
   wsk = new char[80];
   cout << "t a b 1 : " << dopisz_spacje(tab1,wsk) << endl;
   delete [] wsk;
   wsk=NULL;
   cout << "t a b 1 (wersja 2): " << dopisz_spacje2(tab1,&wsk)
                 << endl;
   delete [] wsk;
   cout << "tab2: "; kody(tab2,20);
   cout << "tab3: " << tab3 << endl;
   cout << "wsk1: " << wsk1 << endl;
   cout << "wsk2: " << wsk2 << endl;
return 0;
}
//--------------------
char* dopisz_spacje(const char *wsk1, char *wsk2)
{
```

```
int ile_znakow, i;

   for (ile_znakow=0; *(wsk1+ile_znakow)!=NULL; ile_znakow++)
     /* */;
   for (i=0; i<ile_znakow; i++)
   {
     wsk2[2*i] = wsk1[i];
     wsk2[2*i+1] = ' ';
   }
   wsk2[2*ile_znakow]=NULL;
   return wsk2;
}
//-----------------------
char* dopisz_spacje2(const char *wsk1, char **wsk2)
{
int ile_znakow, i;

   for (ile_znakow=0; *(wsk1+ile_znakow)!=NULL; ile_znakow++)
     /* */;
   *wsk2 = new char[2*ile_znakow+1];
   for (i=0; i<ile_znakow; i++)
   {
     (*wsk2)[2*i] = wsk1[i];
     (*wsk2)[2*i+1] = ' ';
   }
   (*wsk2)[2*ile_znakow]=NULL;
   return *wsk2;
}
//----------------------
void kody(const char *wsk, int rozmiar)
{
int i;
   for (i=0; i<rozmiar; i++)
     cout << int(*(wsk+i)) << ',';
   cout << endl;
}
```

9.11. Wskaźniki do funkcji

Definicja:

```
int (*wsk_do_funkcji)(int, char);
```

oznacza, że `wsk_do_funkcji` jest wskaźnikiem do funkcji wywoływanej z parametrami typu `int` i `char` oraz zwracającej wartość typu `int`.

Nazwa funkcji jest jednocześnie adresem jej początku.

Przykład ❸❶:

```
#include <iostream>
using namespace std;
int dodaj_dwa(int a);
int dodaj_trzy(int a);
int wywolaj(int (*wsk)(int), int liczba);
```

```
//----------------------
int main()
{
int a=10;
int (*wsk_fkc)(int parametr);
int (*tab_wsk_fkc[2])(int); // tablica wskaznikow do funkcji

   cout << "dodaj_dwa: " << dodaj_dwa(a) << endl;
   cout << "dodaj_trzy: " << dodaj_trzy(a) << endl;
   wsk_fkc=dodaj_dwa;
   cout << "wsk. do dodaj_dwa: " << (*wsk_fkc)(a) << endl;
   wsk_fkc=dodaj_trzy;
   cout << "wsk. do dodaj_trzy: " << (*wsk_fkc)(a) << endl;
   cout << "f(dodaj_trzy): " << wywolaj(dodaj_trzy,a) << endl;
   tab_wsk_fkc[0]=dodaj_dwa;
   tab_wsk_fkc[1]=dodaj_trzy;
   cout << "tablica ze wsk. do dodaj_trzy: "
     << (*tab_wsk_fkc[1])(a) << endl;
return 0;
}
//------------------------
int dodaj_dwa(int a)
{  return a+2; }
//------------------------
int dodaj_trzy(int a)
{  return a+3; }
//------------------------
int wywolaj(int (*wsk)(int), int liczba)
{  return (*wsk)(liczba); }
```

---

10. Przeładowanie nazw funkcji

Przeładowanie nazwy funkcji polega na tym, że w danym zakresie ważności jest więcej niż jedna funkcja o tej samej nazwie. Poszczególne funkcje różnią się typami argumentami.

Przykład ❸❷:

```
#include <iostream>
using namespace std;
void drukuj(int);
void drukuj(float);
void drukuj(char);
void drukuj(int, float);
void drukuj(float, int);
//-----------------
int main()
{
int a = 10;
float b = 12.6;
char c = 'a';

   drukuj(a);
   drukuj(b);
   drukuj(c);
```

```
   drukuj(a,b);
   drukuj(b,a);
return 0;
}
//-----------------
void drukuj(int a)
{ cout << a << endl; }
//-----------------
void drukuj(float a)
{ cout << a << endl; }
//-----------------
void drukuj(char a)
{ cout << a << endl; }
//------------------
void drukuj(int a, float b)
{ cout << a << "   " << b << endl; }
//------------------
void drukuj(float b, int a)
{ cout << a << "   " << b << endl; }
```

Przykład – poniższych funkcji nie można przeładować:

```
void funkcja(int tab[]);
void funkcja(int *wsk);
```

10.1. Wskaźnik do funkcji przeładowanej

Przykład:

```
int funkcja(int);
int funkcja(float);
//------------
int (*wsk_do_fkc)(int);
wsk_do_fkc = funkcja; // do wskaznika zostanie przypisany adres
   // funkcji "funkcja(int)" - wynika to z definicji wskaznika
```

11. Klasy

Definicja klasy:

```
class nazwa_klasy
{
   // cialo klasy
   // ..............
} ;  // <<- srednik
```

Definicja obiektu danej klasy (typu zdefiniowanego przez użytkownika):

```
nazwa_klasy zmienna;
```

Definicja wskaźnika i referencji do obiektu danej klasy:

```
nazwa_klasy * wsk ;
nazwa_klasy & refer = zmienna;
```

11.1. Składniki klasy

Dostęp do składników klasy:

- obiekt.składnik
- wskaźnik -> składnik
- referencja.składnik

Przykład ❸❸:

```
#include <iostream>
#include <string>
using namespace std;
//---------------------
class COsoba
{
public:
   char m_sImie[80];
   char m_sNazwisko[80];
   int m_nWaga;
   int m_nWzrost;
};
//---------------------
int main()
{
COsoba Kowalski;
COsoba &rOsoba = Kowalski; // referencja (musi byc
                  //           zainicjalizowana)
COsoba *pOsoba; // wskaznik

   strcpy(Kowalski.m_sImie, "Jan");
   strcpy(Kowalski.m_sNazwisko, "Kowalski");
   Kowalski.m_nWaga = 65;
   Kowalski.m_nWzrost = 178;
   pOsoba = & Kowalski;

   cout << Kowalski.m_sImie << " " << Kowalski.m_sNazwisko
          << endl;
   cout << "waga: " << pOsoba->m_nWaga << endl;
   cout << "wzrost: " << rOsoba.m_nWzrost << endl;
return 0;
}
```

11.2. Funkcje składowe klasy

Definicja funkcji składowej klasy jako funkcji typu `inline`:

```
class nazwa_klasy
{
   // ............
   typ nazwa_funkcji(lista_parametrow)
     {
     // cialo funkcji
     }
};
```

Deklaracja i definicja funkcji składowej klasy:

```
class nazwa_klasy
{
```

```
    // ............
    typ nazwa_funkcji(lista_parametrow);
};
typ nazwa_klasy :: nazwa_funkcji(lista_parametrow)
{
    // cialo funkcji
}
```

Przykład ❸❹:

```
#include <iostream>
#include <string>
using namespace std;
//---------------------
class COsoba
{
public:
    char m_sImie[80];
    char m_sNazwisko[80];
    int m_nWaga;
    int m_nWzrost;
    void WstawDane(char *imie, char *nazwisko,
                   int waga, int wzrost);
    void WydrukujDane();
};
void COsoba::WstawDane(char *imie, char *nazwisko,
                       int waga, int wzrost)
{
    strcpy(m_sImie, imie);
    strcpy(m_sNazwisko, nazwisko);
    m_nWaga = waga;
    m_nWzrost = wzrost;
}
void COsoba::WydrukujDane()
{
    cout << m_sImie << " " << m_sNazwisko << endl;
    cout << "waga: " << m_nWaga << endl;
    cout << "wzrost: " << m_nWzrost << endl;
}
//---------------------
int main()
{
COsoba Kowalski;
COsoba *pOsoba;

    Kowalski.WstawDane("Jan","Kowalski",65,178);
    pOsoba = & Kowalski;
    pOsoba->WydrukujDane();
return 0;
}
```

11.3. Rodzaje dostępu do składników klasy

Rodzaje dostępu do składników (zmiennych i funkcji) klasy:

- `private` – składniki dostępne są wyłącznie dla składników klasy,
- `public` – składniki są dostępne bez ograniczeń.

Przykład:
```
class nazwa_klasy
{
// skladniki prywatne
private:
   // deklaracje/definicje składników
// ---------------------
// składniki publiczne
public:
   // deklaracje/definicje składników
};
```
Dopóki w definicji klasy nie wystąpi żadna etykieta to składniki mają dostęp `private`.

11.4. Obiekty będące składnikami klasy

Przykład klasy zawierającej obiekt innej klasy:
```
class zarowka
{
public:
   int moc;
   double srednica;
   void zaswiec();
   void zgas();
};
class lampa
{
public:
   int wysokosc;
   zarowka punkt_swietlny1;
   zarowka punkt_swietlny2;
   void zaswiec();
   void zgas();
};
```

11.5. Przesyłanie do funkcji argumentów będących obiektami

Przykład ❸❺:
```
#include <iostream>
#include <string>
using namespace std;
//---------------------
class COsoba
{
public:
   char m_sImie[80];
   char m_sNazwisko[80];
   int m_nWaga;
   int m_nWzrost;
   void WstawDane(char *imie, char *nazwisko,
                     int waga, int wzrost);
};
void COsoba::WstawDane(char *imie, char *nazwisko,
                        int waga, int wzrost)
{
   strcpy(m_sImie, imie);
   strcpy(m_sNazwisko, nazwisko);
```

```
   m_nWaga = waga;
   m_nWzrost = wzrost;
}
//---------------------
void wydrukuj_dane(COsoba osoba);
void wydrukuj_nazwisko(COsoba & osoba);
//----------------------
int main()
{
COsoba Kowalski;

   Kowalski.WstawDane("Jan","Kowalski",65,178);
   wydrukuj_dane(Kowalski);
   wydrukuj_nazwisko(Kowalski);
return 0;
}
//---------------------
void wydrukuj_dane(COsoba osoba)
// przekazanie obiektu przez wartosc
{
   cout << osoba.m_sImie << " " << osoba.m_sNazwisko << endl;
   cout << "waga: " << osoba.m_nWaga << endl;
   cout << "wzrost: " << osoba.m_nWzrost << endl;
}
//----------------------
void wydrukuj_nazwisko(COsoba & osoba)
// przekazanie obiektu przez referencje
{
   cout << "Nazwisko: " << osoba.m_sNazwisko << endl;
}
```

11.6. Składnik statyczny

```
class nazwa_klasy
{
// cialo klasy
   static typ1 nazwa1;     // deklaracja składnika statycznego
   static typ2 nazwa2;
// ...........
};
typ1 nazwa_klasy::nazwa1;  // definicja składnika statycznego
typ2 nazwa_klasy::nazwa2 = wartosc; // definicja
                           // i inicjalizacja składnika stat.
```

Cechy składnika statycznego:

- miejsce w pamięci dla składnika statycznego jest przydzielane tylko raz, niezależnie od ilości obiektów danej klasy – składnik statyczny jest wspólny dla wszystkich obiektów klasy,
- składnik statyczny istnieje nawet wtedy gdy nie ma żadnego obiektu danej klasy,
- dostęp do składnika statycznego:
  `nazwa_klasy::nazwa_skladnika`
- dostęp do składnika statycznego gdy istnieją obiekty danej klasy:
  `obiekt.nazwa_skladnika`
  `wskaznik->nazwa_skladnika`

11.7. Statyczna funkcja składowa

```
class nazwa_klasy
{
// ciało klasy
   static typ nazwa_funkcji(parametry); // deklaracja funkcji
// ...
};
```

Cechy statycznej funkcji składowej:

- funkcję można wywołać gdy nie istnieje żaden obiekt danej klasy,
- wewnątrz funkcji nie ma wskaźnika `this`,
- funkcja nie ma możliwości odwoływania się do niestatycznych składników klasy,
- sposoby wywołania funkcji: takie jak dostęp do składnika statycznego.

Przykład ❸❻:

```
#include <iostream>
using namespace std;
class klasa
{
public:
   int numer;
   static long licznik;
   static void zerujlicznik(){ licznik=0; }
};
long klasa::licznik = 0;
//---------
int main()
{
   cout << "licznik=" << klasa::licznik << endl;
   klasa obiekt1;
   obiekt1.licznik++;
   cout << "licznik=" << klasa::licznik << endl;
   klasa obiekt2;
   obiekt2.licznik++;
   cout << "licznik=" << klasa::licznik << endl;
   klasa::zerujlicznik();
   cout << "licznik=" << klasa::licznik << endl;
return 0;
}
```

11.8. Funkcje składowe typu const

```
class nazwa_klasy
{
// ciało klasy
   typ nazwa_funkcji(parametry) const;
// ............
};
typ nazwa_klasy::nazwa_funkcji(parametry) const
{
   // ciało funkcji
}
```

Są to funkcje deklarujące, że nie będą zmieniały obiektu, na rzecz którego pracują.
Służą do pracy z obiektami zdefiniowanymi jako `const`.

12. Funkcje zaprzyjaźnione

Są to funkcje, które mają dostęp do składników prywatnych danej klasy.

```
class nazwa_klasy
{
   friend typ nazwa_funkcji(parametry);
// cialo klasy
};
typ nazwa_funkcji(parametry)
{
   // cialo funkcji
}
```

Funkcje te nie posiadają wskaźnika `this` do składników klasy, z którą są zaprzyjaźnione, a więc muszą posługiwać się operatorami **.** (kropka) lub **->**.

Przykład ❸❼:

```
#include <iostream>
using namespace std;
class klasa
{
   friend int przyjaciel(klasa ob);
private:
   int liczba;
public:
   void ustaw(int wartosc) { liczba=wartosc; }
   void wyswietl() { cout << "liczba=" << liczba << endl; }
};
//---------------
int przyjaciel(klasa ob)
{
   return ob.liczba;
}
//---------------
int main()
{
klasa objekt;

   objekt.ustaw(10);
   objekt.wyswietl();
   cout << "Przyjaciel: liczba=" << przyjaciel(objekt) << endl;
return 0;
}
```

13. Struktury, unie i pola bitowe

13.1. Struktura

Struktura to klasa, w której wszystkie składniki są publiczne. Składnikami struktur nie mogą być funkcje.

```
struct nazwa_struktury
{
// lista składnikow
};
```

13.2. Unia

Unia to struktura, w której poszczególne składniki zajmują to samo miejsce w pamięci.
Przykład:

```
union pojemnik
{
   char c;
   int i;
   float f;
};
```

13.3. Pole bitowe

Pole bitowe to typ składnika klasy polegający na tym, że informacja jest przechowywana na określonej liczbie bitów.
Przykład:

```
class klasa
{
public:
   unsigned int bit1 : 1;
   unsigned int bit4 : 4;
};
```

---

14. Konstruktory i destruktory

14.1. Konstruktor

Konstruktor to funkcja składowa klasy postaci

`nazwa_klasy(*parametry*);`

która jest <u>automatycznie</u> wywoływana podczas definiowania obiektu danej klasy.
Cechy konstruktora:

- konstruktor może być przeładowywany,
- konstruktor nie ma wyspecyfikowanego typu wartości zwracanej,
- konstruktor nie zwraca żadnej wartości (nawet `void`),
- konstruktor może być wywoływany dla tworzenia obiektów z modyfikatorem `const`, ale sam nie może być funkcją typu `const`,
- konstruktor nie może być typu `static`,
- nie można posłużyć się adresem konstruktora.

14.2. Kiedy wywoływany jest konstruktor

14.2.1. Konstruowanie obiektów lokalnych

Obiekty lokalne automatyczne – konstruktor uruchamiany jest w momencie, gdy program napotyka definicję obiektu.

Obiekty lokalne statyczne – konstruktor zostanie uruchomiony w momencie uruchomienia programu.

14.2.2. Konstruowanie obiektów globalnych

Konstruktor zostanie uruchomiony w momencie uruchomienia programu.

14.2.3. Konstrukcja obiektów tworzonych operatorem new

Konstruktor zostanie uruchomiony po wywołaniu operatora `new`. Konstruktor nie przydziela miejsca w pamięci dla tworzonego obiektu – jedynie inicjalizuje wartości składników.

14.2.3. Inne przypadki uruchamiania konstruktora

Konstruktor wywoływany jest podczas tworzenia obiektów chwilowych swojej klasy.
Konstruktor jest wywoływany jeśli jest tworzony obiekt jakiejś klasy, który zawiera obiekt klasy tego konstruktora.

14.3. Destruktor

Destruktor to funkcja składowa klasy postaci

```
~nazwa_klasy();
```

która jest automatycznie wywoływana podczas likwidowania obiektu danej klasy.
Cechy destruktora:

- destruktor nie może zwracać żadnej wartości (nawet `void`),
- destruktor jest wywoływany bez jakichkolwiek argumentów,
- destruktor nie może być przeładowany,
- nie można pobrać adresu destruktora,
- destruktor nie może być funkcją typu `const`, ale może pracować na rzecz obiektów typu `const`.

Przykład ❸❽:

```
#include <iostream>
#include <string>
using namespace std;
//-------------------
class napis
{
private:
   char tekst[80];
   char przypis[80];
   int x;
   int y;
public:
   napis();
   napis(char *t, char *p, int wspx=0, int wspy=0);
   ~napis();
   void wyswietl();
};
//-------
napis::napis()
{
   strcpy(tekst,"BRAK");
   strcpy(przypis,"BRAK");
   x=0; y=0;
   wyswietl();
}
//--------
napis::napis(char *t, char *p, int wspx, int wspy)
{
   strcpy(tekst,t);
   strcpy(przypis,p);
```

```
      x=wspx;
      y=wspy;
      wyswietl();
   }
   //---------
   napis::~napis()
   {
      cout << "destruktor: " << tekst << endl;
   }
   //---------
   void napis::wyswietl()
   {
      cout << "\n------------------\n";
      cout << "tekst: " << tekst << endl;
      cout << "przypis: " << przypis << endl;
      cout << "x=" << x << "   y=" << y << endl;
      cout << "\n------------------\n";
   }
   //--------------------------------------
   int main()
   {
      napis Tekst1("Tekst1","Przypis1",10,10);
      napis Tekst2;
      napis Tekst3("Tekst3","Przypis3");
      napis Tekst4 = napis("Tekst4","Przypis4",1,2);
      napis *wsk;
      wsk = new napis("Wskaznik","Przypis",5,3);
      delete wsk;
   return 0;
   }
```

14.4. Konstruktor domniemany

Konstruktor domniemany ta taki konstruktor, który można wywołać bez żadnego argumentu.
Przykład:

```
   class klasa1
   {
   //...........
   public:
      klasa1(int a);
      klasa1();    // konstruktor domniemany
      klasa1(float b);
   //.........
   };
   class klasa2
   {
   //...........
   public:
      klasa2(int a);
      klasa2(float b);
      klasa2(int a=1; double b=3.1); // konstruktor domniemany
   //............
   };
```

Jeśli klasa nie ma żadnego konstruktora, to kompilator sam wygeneruje dla tej klasy konstruktor domniemany.

14.5. Lista inicjalizacyjna konstruktora

Definicja konstruktora z listą inicjalizacyjną:

```
nazwa_klasy::nazwa_klasy(parametry) : lista_inicjalizacyjna
{
   // ciało konstruktora
}
```

Etapy wykonania konstruktora:

1. inicjalizacja składników z listy inicjalizacyjnej konstruktora,
2. przypisania i inne akcje zdefiniowane w ciele konstruktora

Cechy listy inicjalizacyjnej:

- składnik bez modyfikatora `const` można inicjalizować przez listę inicjalizacyjną lub przez przypisanie w ciele konstruktora,
- składnik typu `const` można inicjalizować tylko za pomocą listy inicjalizacyjnej,
- lista inicjalizacyjna nie może inicjalizować składnika `static`.

Przykład ❸❾:

```
#include <iostream>
#include <string>
using namespace std;
//-----------------
class klasa
{
public:
   int a;
   int b;
   const int c;
   char tekst[20];
   klasa(int aa, int bb, int cc, char *t);
   void wyswietl();
};
klasa::klasa(int aa, int bb, int cc, char *t)
           : a(aa), b(bb), c(cc)
{
   strcpy(tekst,t);
   wyswietl();
}
void klasa::wyswietl()
{
   cout << "--------------\n";
   cout << tekst << endl;
   cout << "a=" << a << " b=" << b << " c=" << c << endl;
   cout << "--------------\n";
}
//--------------------
int main()
{
   klasa dane(1,2,3,"Dane");
return 0;
}
```