1.	Programowanie strukturalne i obiektowe
2.	Typ wyliczeniowy, lista wyliczeniowa. Rzutowanie w C++. Wskaźniki i referencje. Działania na wskaźnikach. Wskaźniki typu const. Tablice wskaźników. Tworzenie elementów na stercie i zwalnianie pamięci.
3-4.	Funkcje i operatory. Wywołanie funkcji. Modyfikator const. Przekazywanie argumentów (wartość, wskaźnik, referencja) i zwracane wartości. Przeciążanie funkcji i operatorów. Szablony funkcji. Makro. Argumenty funkcji main(). Funkcje inline.
5-7.	Klasa, struktura, unia. Obiekty. Składowe i funkcje składowe klasy. Struktura programu przy stosowaniu programowania obiektowego. Dostęp do składowych i funkcji klasy. Umiejscowienie funkcji składowych klasy. Składowe statyczne klasy i ich definicja.
8-10.	Tworzenie obiektów klasy. Konstruktor. Umiejscowienie konstruktora. Użycie wielu konstruktorów. Likwidacja obiektu klasy. Umiejscowienie destruktora. Tablice obiektów. Klasy zagnieżdżone
11-12	Wskaźniki do obiektów klasy. Tworzenie i likwidacja. obiektów klasy przy użyciu wskaźników. Dostęp do składowych i funkcji składowych klasy za pomocą wskaźników. Wskaźnik this.
13-14.	Działania na obiektach. Przeciążanie funkcji i operatorów. Rzutowanie.
15-18.	Dziedziczenie. Typy dziedziczenia i dostęp do składowych i funkcji klasy. Konstruktory i destruktory. Dostęp do składowych i funkcji składowych klasy bazowej i pochodnej. Przesłanianie i ukrywanie metod klasy bazowej. Rzutowanie .
19-22	Dziedziczenie wielokrotne. Konstruktory i destruktory przy dziedziczeniu wielokrotnym. Eliminacja niejednoznaczności. Dziedziczenie ze wspólnej klasy bazowej. Funkcje wirtualne i klasy abstrakcyjne. Rzutowanie.
23-28.	STL
29-30.	Zaliczenie

Poniższy kod:

```
struct Struct
{int val;};
int main()
{Struct.val = 5;       return 0;}
```

a) przypisze zmiennej val wartość 5
b) nie zostanie wykonany - w strukturze Struct nie ma konstruktora
c) nie zostanie wykonany - dostęp do zmiennej val jest private
d) nie zostanie wykonany - nie został stworzony obiekt struktury Struct

Efektem poniższego programu:

```
class Ptak{public: virtual void Glos() const=0;};
class Wrona: public Ptak {public:void Glos()
   const{cout<<"Kra"<<endl;}};
int main()
{Ptak ptak; Wrona wrona; wrona.Glos(); return 0;}
```

będzie:
a) „Kra”
b) Błąd polegający na braku konstruktora klasy Ptak
c) błąd polegający na próbie stworzenia obiektu klasy abstrakcyjnej
d) Błąd polegający na braku konstruktora klasy Wrona

Poniższy kod:

```
class Class
{int val;};
int main()
{class Class; Class.val = 5;      return 0;}
```

a) przypisze zmiennej val wartość 5
b) nie zostanie wykonany - dostęp do zmiennej val jest private
c) nie zostanie wykonany - nie został stworzony obiekt klasy Class
d) nie zostanie wykonany - w klasie Class nie ma konstruktora

Definicja funkcji składowej Function poza ciałem klasy Class musi mieć postać

a) typ_zwracany Class.Function(){}
b) typ_zwracany Class::Function(){}
c) typ_zwracany Class::Function(){} const
d) Funkcji składowej nie wolno definiować poza ciałem klasy

Programowanie strukturalne

- Program jest zestawem zadań do wykonania
- Jest rozumiany jako seria procedur, działających na danych
- Procedura (funkcja, metoda) – ciąg następujących po sobie instrukcji
- Zadania skomplikowane są dzielone na prostsze, aż do momentu, kiedy wszystkie są zrozumiałe

Cechy charakterystyczne :

1. Dane są odseparowane od procedur i nie ma możliwości ich połączenia
2. Programista musi pamiętać, która funkcja wywołuje inne funkcje i które dane są zmieniane
3. Powtarzanie fragmentów kodu
4. Koncentruje się na czynnościach
5. Problemy z połączeniem zmiennych różnych typów
6. Problemy z modyfikacjami kodu i rozbudową programu

Przykład: Obliczenie średniego wynagrodzenia pracownika

1. Policzenie liczby pracowników
2. Obliczenie zarobków poszczególnych osób (odczytanie danych każdego pracownika <otwarcie pliku danych, odszukanie właściwych danych, odczyt danych>, dodanie jego pensji do sumy, przejście do kolejnej osoby)
3. Zsumowanie wszystkich pensji
4. Podzielenie sumy przez liczbę pracowników

Programowanie zorientowane obiektowo

1. Jego zadaniem jest modelowanie obiektów (rzecz lub koncepcja), a nie danych
2. Określenie właściwości obiektu (atrybut, funkcja)
3. Zdefiniowanie czynności wykonywanych przez obiekt
4. Określenie związków między obiektami

Cechy programowania zorientowanego obiektowo

1. Funkcjonalność - program musi być zawsze zorientowany na klienta.
2. Elastyczność – programista musi przewidywać zmieniające się warunki eksploatacji oprogramowania i wymagania klienta
3. Wzorce projektowe – wielokrotne wykorzystanie tego samego kodu
4. Hermetyzacja – używana w celu oddzielenia fragmentów kodu, które ulegają zmianom od tych, które pozostają takie same

Podstawy programowania zorientowanego obiektowo

Hermetyzacja (ukrywanie danych) – wszystkie właściwości obiektu są zgrupowane w jego wnętrzu – obiekt jest traktowany jak „czarna skrzynka”. Oddziela dane od aplikacji.

C++ obsługuje hermetyzację poprzez możliwość tworzenia typów danych zdefiniowanych przez użytkownika – KLASY.

Użytkownicy dobrze zdefiniowanej klasy nie mają wpływu na jej wewnętrzne działanie – potrafią tylko z niej korzystać.

Podstawy programowania zorientowanego obiektowo

Dziedziczenie (w celu ponownego wykorzystania)

C++ umożliwia deklarowanie nowych typów stanowiących rozszerzenie typów już zadeklarowanych.

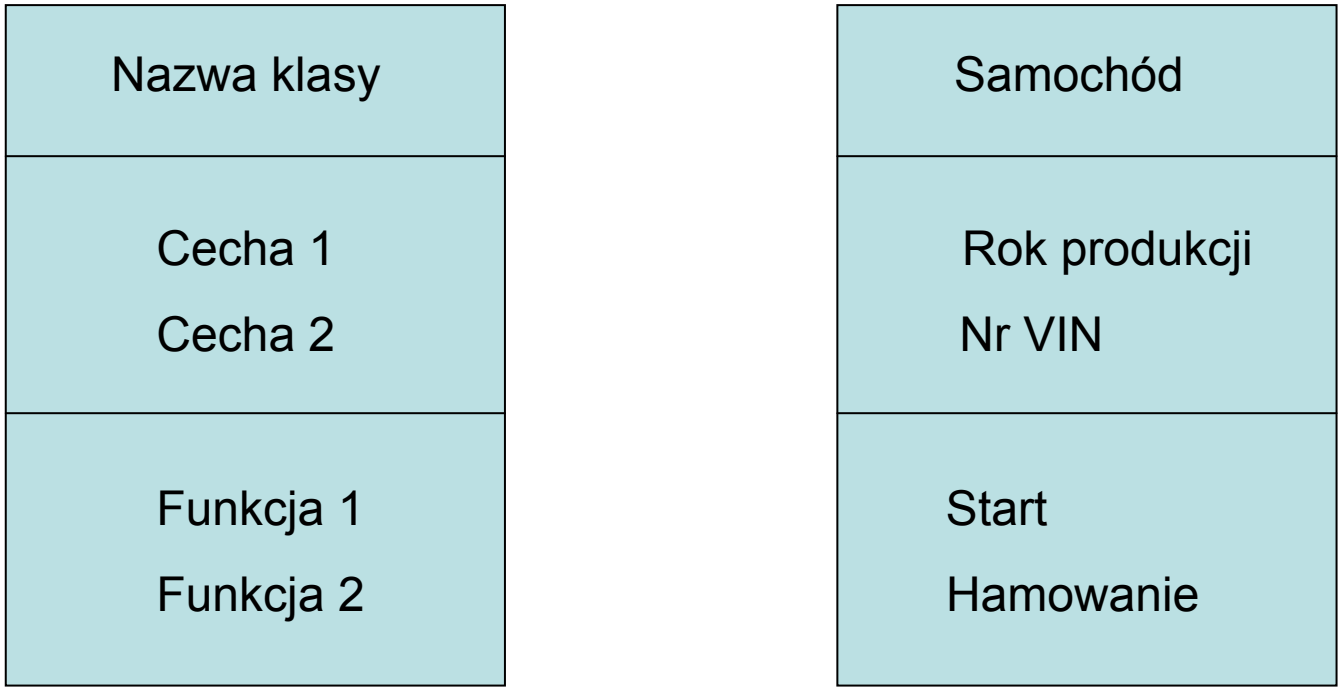
Polimorfizm – różne obiekty wykonują podobne czynności

W C++ polimorfizm klas i polimorfizm obiektów

Klasa - grupa zmiennych, często różnych typów wraz z zestawem funkcji (czynności) odnoszących się do nich.

Obiekt - indywidualny element klasy

UML – Unified Modelling Language



Typ wyliczeniowy

```
// Typ wyliczeniowy 01
#include<iostream>
using namespace std;

enum Kolor
{ Trefl, Karo, Kier, Pik };

int main()
{
        Kolor karta_1=Pik;
        Kolor karta_2=Karo;
        Kolor karta_3=Trefl;
        Kolor karta_4=Kier;

        cout<<karta_1<<"\t"<<karta_2<<"\t"
                <<karta_3<<"\t"<<karta_4<<endl;
        cout<<sizeof(karta_1)<<endl;

        system("pause");
        return 0;
}
```

```
 3       1       0       2
 4
 Press any key to continue . . .
```

```
// Tablice wyliczeniowe
#include<iostream>
using namespace std;

enum Karta
{
        Dwojka=2,
        Trojka,
        Czworka,
        Piatka,
        Szostka,
        Siodemka,
        Osemka,
        Dziewiatka,
        Dziesiatka,
        Walet,
        Dama,
        Krol,
        As};
```

```
void main()
{
        Karta Losuj[14]=
        {
                Dwojka,
                Trojka,
                Czworka,
                Piatka,
                Szostka,
                Siodemka,
                Osemka,
                Dziewiatka,
                Dziesiatka,
                Walet,
                Dama,
                Krol,
                As
        };
```

```
        //Losuj[14]=Losuj[3]+Losuj[6];

        for (int i=0; i<14 ; i++)
        {cout<<Losuj[i]<<endl;}

        int wynik=Losuj[3]+Losuj[6];
        cout<<wynik<<endl;

        system("pause");
        return;
}
```

```
2
3
4
5
6
7
8
9
10
11
12
13
14
0
13
Press any key to continue . . .
```

Rzutowanie w C++

Mechanizm pozwalający na tymczasową lub stałą zmianę interpretacji obiektu przez kompilator – nie powoduje rzeczywistej zmiany samego obiektu. Wymusza na kompilatorze interpretację obiektu w sposób przewidziany przez programistę.

Dostępne operatory rzutowania:

1. `static_cast`
2. `dynamic_cast`
3. `reinterpret_cast`
4. `const_cast`

```
typ_docelowy wynik =
  typ_rzutowania<typ_docelowy>(obiekt_do_rzutowania);
```

`reinterpret_cast` – konwersja wskaźników między dowolnymi, nawet niepowiązanymi typami danych – wymusza na kompilatorze dowolne rzutowanie - NIEZALECANE.

`const_cast` – wyłączenie modyfikatora dostępu `const` w obiekcie – użycie może doprowadzić do nieprzewidzianego zachowania obiektu.

```
// Rzutowanie static_cast_1

#include<iostream>
using namespace std;

enum Karta
{
        Dwojka=2,
        Trojka,
        Czworka,
        Piatka,
        Szostka,
        Siodemka,
        Osemka,
        Dziewiatka,
        Dziesiatka,
        Walet,
        Dama,
        Krol,
        As};
```

```
void main()
{
	Karta Losuj[15]=
	{
		Dwojka,
		Trojka,
		Czworka,
		Piatka,
		Szostka,
		Siodemka,
		Osemka,
		Dziewiatka,
		Dziesiatka,
		Walet,
		Dama,
		Krol,
		As
	};
	Losuj[13]=static_cast<Karta>(Losuj[3]+Losuj[6]);
	Losuj[14]=static_cast<Karta>(Losuj[3]-Losuj[6]);
```

```
        for (int i=0; i<15 ; i++)
        {cout<<Losuj[i]<<endl;}

        system("pause");
        return;
}
```

```
2
3
4
5
6
7
8
9
10
11
12
13
14
13
-3
Press any key to continue . . .
```

```
// Rzutowanie static_cast

#include<iostream>
using namespace std;

int main()
{
        char napis[15]={"ABCDEFGH"};
        int a=napis[0];
        cout<<a<<endl;
        napis[9]=char(a+a+1);
        cout<<napis[9]<<endl;
        cout<<"Uzycie static_cast\n";
        a=static_cast<int>(napis[0]);
        napis[10]=static_cast<char>(2*a+1);
        cout<<napis[10]<<endl;

        system("pause");
        return 0;
}
```

```
65
â
Uzycie static_cast
â
Press any key to continue . . .
```

```
// Rzutowanie dynamic_cast

#include<iostream>
using namespace std;

int main()
{
	double pi=3.1415;
	double e=2.71;
	int suma=pi+e;
	cout<<suma<<endl;
	/*int suma1=dynamic_cast<int>(e)+dynamic_cast<int>(pi);
	cout<<suma1<<endl;*/

	system("pause");
	return 0;
}



5
Press any key to continue . . .
```

```
1>dynamic.cpp
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_01_05\w_01_05\dynamic.cpp(14) : error C2680: 'int' : invalid target type for
dynamic_cast

1>       target type must be a pointer or reference to a defined class
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_01_05\w_01_05\dynamic.cpp(15) : error C2680: 'char' : invalid target type for
dynamic_cast
1>       target type must be a pointer or reference to a defined class

1>Build log was saved at "file://e:\Praca\Dydaktyka\Programowanie obiektowe\Visual
Project\Wykład 2016\W_01_05\W_01_05\Debug\BuildLog.htm"
1>W_01_05 - 2 error(s), 0 warning(s)
========== Build: 0 succeeded, 1 failed, 0 up-to-date, 0 skipped ==========
```

```
// Rzutowanie static_cast

#include<iostream>
using namespace std;

int main()
{
        double pi=3.1415;
        double e=2.71;
        int suma=pi+e;
        cout<<suma<<endl;
        int suma1=reinterpret_cast<int>(e)+reinterpret_cast<int>(pi);
        cout<<suma1<<endl;

        system("pause");
        return 0;
}
```

```
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład 2016\w_01_07\w_01_07\static.cpp(12) : error C2440: 'reinterpret_cast' : cannot convert from 'double' to 'int'
1>      Conversion is a valid standard conversion, which can be performed implicitly or by use of static_cast, C-style cast or function-style cast
```

```
// Rzutowanie const_cast

#include<iostream>
using namespace std;

int main()
{
	const double pi=3.1415;
	//pi+=2;
	double pi_1=const_cast<double>(pi);

	cout<<pi<<endl;

	system("pause");
	return 0;
}
```

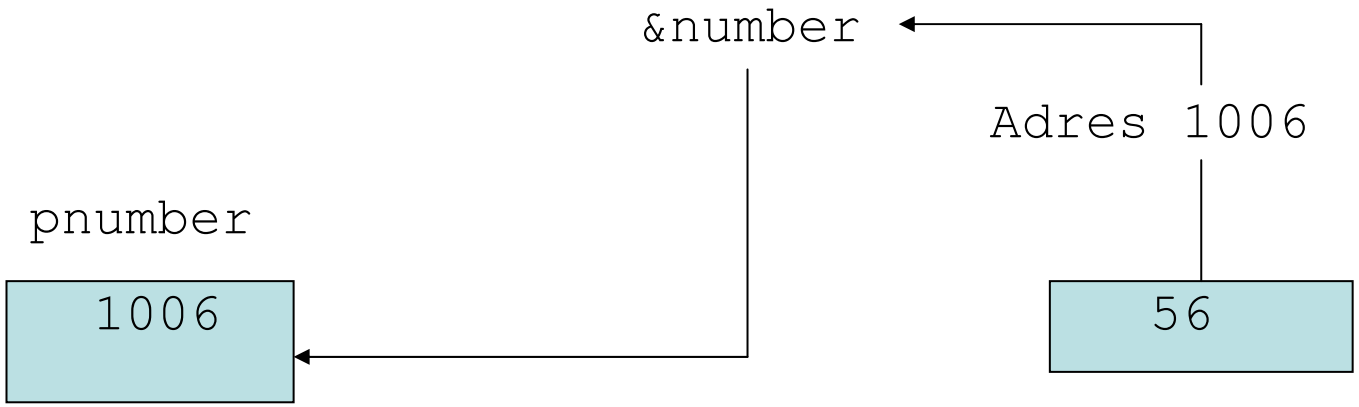
```
1>e:\praca\dydaktyka\programowanie obiektowe\visual project\wykład
2016\w_01_09\w_01_09\const.cpp(10) : error C2440: 'const_cast' : cannot
convert from 'const double' to 'double'
1>	Conversion is a valid standard conversion, which can be performed
implicitly or by use of static_cast, C-style cast or function-style cast
```

Wskaźniki

Zmienna przechowująca dane o adresie innej zmiennej określonego typu.

Deklarowanie wskaźnika w C++

```
long *pnumber long* pnumber
long *pnumber;
number = 56;
pnumber = &number
```



```
// Użycie wskaźników
#include <iostream>
using namespace std;

int main()
{
long *pnumber=NULL; //Deklaracja inicjalizacja wskaźnika
long number1=44, number2=88; pnumber=&number1; //adres do
wskaŹnika
*pnumber+=11;       //number1 + 11
cout << endl << " number1 = " << number1
 << " pnumber = " << hex << pnumber;

pnumber = &number2; //zmiana adresu
number1 = *pnumber*10; //number2 * 10
cout << endl;
cout<< " number1 = " << dec << number1
       << " pnumber = " << hex << *pnumber
       << " *pnumber = " << dec <<*pnumber << endl;

system("pause");
return 0;
}
```

```
 number1 = 55 pnumber = 0012FF54
 number1 = 880 pnumber = 58 *pnumber = 88
Aby kontynuować, naciśnij dowolny klawisz . .
.
```

```
// Użycie wskaźników

#include <iostream>

using namespace std;

int main()
{

        double zmienna_double=4.6;
        double *pdouble=0;
        pdouble=&zmienna_double;
        cout << pdouble <<endl;
        cout << *pdouble <<endl;
        pdouble=new double(23.5);
        cout << pdouble <<endl;
        cout << *pdouble <<endl;
        double nowa_zmienna=*pdouble;
        cout <<nowa_zmienna<<endl;
        pdouble=new double;
```

```
cout << pdouble <<endl;
       cout << *pdouble <<endl;
       *pdouble= 34.56;
       cout << pdouble <<endl;
       cout << *pdouble <<endl;
       cout<<endl;

       system ("pause");
       return 0;

}
```

```
0024FB68
4.6
00784FB8
23.5
23.5
00785000
-6.27744e+066
00785000
34.56

Press any key to continue . . .
```