

# Laboratorium1 – podstawowe różnice między C a C++

## Pierwszy program.

```
#include <iostream>

using namespace std;

int main(void)
{
    cout << "Witaj C++" << endl; // drukuje napis
    return 0;
}
```

## Zadanie 1

Stwórz projekt. Przepisz i uruchom powyższy program. Program uruchom również w debuggerze i wykonaj go krok po kroku.

## Wejście wyjście

W C++ do obsługi wejścia służą strumienie. Dwa najbardziej podstawowe to `cin` pobierania danych ze standardowego wejścia oraz `cout` do wyświetlania danych na standardowym wyjściu. Kolejnym często używanym strumieniem jest `cerr` służący do wysyłania komunikatów na standardowe wyjście błędów. Wymienione strumienie zdefiniowane są w pliku nagłówkowym `iostream`.

Dane wczytuje się w następujący sposób

```
int a, b, c;
cin >> a;
cin >> b >> c;
```

Wyświetlanie danych odbywa się analogicznie. Manipulator `endl` wysyła znak końca linii do strumienia.

```
cout << "a= " << a << endl;
cout << "b= " << b << " c= " << c << endl;
```

Podobnie jak w przypadku funkcji `printf()/scanf()` z języka C również w przypadku strumieni można określać format wyświetlanych/wczytywanych danych. Służą do tego manipulatory zdefiniowane w pliku nagłówkowym `ios` oraz `iomanip`.

Manipulator	Działanie
<code>setprecision(precyzja)</code>	ustawia precyzję (liczbę miejsc po przecinku) wyświetlanych/wczytywanych liczb
<code>setfill(znak)</code>	Zmienia znak wypełniający
<code>setw(szerokosc)</code>	Ustawia szerokość następnego pola
<code>left</code>	Ustawia wyrównanie do lewej
<code>right</code>	Ustawia wyrównanie do prawej
<code>dec</code>	Zmienia podstawę systemu liczbowego na 10
<code>hex</code>	Zmienia podstawę systemu liczbowego na 16

oct	Zmienia podstawę systemu liczbowego na 8
setbase(podstawa)	Zmienia podstawę systemu liczbowego na podstawę. Zero powoduje, że podstawa będzie rozpoznawana automatycznie na podstawie formatu (0, 0x, itd.)
showbase	Włącza wyświetlanie podstawy systemu liczbowego
noshowbase	Wyłącza wyświetlanie podstawy systemu liczbowego
fixed	Włącza stałą liczbę cyfr po przecinku dla liczb zmiennoprzecinkowych
scientific	Włącza reprezentację naukową dla liczb zmiennoprzecinkowych
resetioflags(flagi)	Zeruje podaną flagę np. ios_base::scientific

Użycie manipulatorów jest bardzo proste. Wystarczy wysłać je do strumienia. Przykład poniżej powoduje wyświetlenie liczby 114 w systemie szesnastkowym i w formacie pozwalającym określić podstawę systemu liczbowego.

```
cout << showbase << hex << 114 << endl;
```

## Zadanie 2

Napisz konwerter liczb całkowitych pozwalający na konwersję między zapisem w systemie dziesiętnym a ósemkowym i szesnastkowym. Konwerter powinien pozwalać na wybór reprezentacji danych wyświetlanych.

### Typ string

Jednym z mankamentów języka C był brak typu łańcuchowego. C++ uzupełnia ten brak za pomocą typu (i klasy) string. Jest on w pełni kompatybilny ze strumieniami.

```
string napis;
cout << "Podaj napis: " << endl;
cin >> napis;
cout << "Napis to: " << napis << ", jego długość to: " << napis.length()
    << ", a pierwszy znak to: " << napis[0] << endl;
```

Oprócz standardowych strumieni cin i cout istnieje również możliwość tworzenia strumieni związanych z łańcuchami. Pozwalają one na odczyt i zapis. Można np. z ich pomocą uzyskać łańcuch sformatowany za pomocą manipulatorów (**utworzony łańcuch (string) można odczytać za pomocą metody str()**) a także przetwarzać łańcuchy.

```
stringstream strumien;
strumien << "10 20 30" << " napis";
cout<<strumien.str()<<endl;
int a, b, c;
strumien >> a >> b >> c;
cout << a << " " << b << " " << c << endl;
string napis;
strumien >> napis;
cout << napis << endl;
```

## Zadanie 3

Napisz program wczytujący dane pracowników do tablicy struktur. Struktura zawiera następujące dane: imię, nazwisko, stawkę godzinową, liczbę godzin pracy, datę zatrudnienia. Po wczytaniu program wyświetla dane w postaci tabeli (wyjście programu należy sformatować tak aby kolumny tabeli były wyrównane. Napisz funkcję `string toString(Pracownik)`, która konwertuje dane pracownika zapisane w strukturze do postaci tekstowej.

### Przestrzenie nazw

Czasami (szczególnie w rozbudowanych projektach) wygodnie jest nadać kilku funkcjom czy zmiennym taką samą nazwę. Dlatego wprowadzono przestrzenie nazw. W obrębie jednej przestrzeni nazwy muszą być unikalne (nie licząc przeciążania funkcji) jednak w różnych przestrzeniach nazwy mogą się powtarzać. W poprzednich przykładach stosowana była przestrzeń nazw `std`.

Domyślną przestrzeń można ustawić za pomocą:

```
using namespace std;
```

można też wybrać niektóre elementy z przestrzeni:

```
using std::cout;  
using std::endl;
```

Jeżeli nie zastosujemy dyrektywy `using` a chcemy się odwołać do elementu należącego do jakiejś przestrzeni konieczne jest stosowanie nazw funkcji, zmiennych, obiektów itd. w postaci

```
std::cout << "Witaj C++" << std::endl;
```

Przestrzenie i ich elementy można definiować w następujący sposób:

```
namespace przestrzen  
{  
    typ zmienna;  
    void funkcja()  
    {  
        //treść funkcji  
    }  
}  
  
namespace inna_przestrzen  
{  
    typ zmienna;  
    void funkcja();  
}  
  
void inna_przestrzen::funkcja()  
{  
    //treść funkcji  
    zmienna=1;  
    przestrzen::zmienna=2;  
}
```

## Zadanie 4

Napisz program, w którym znajdują się dwie przestrzenie nazw: `pies` i `kot`.

- W każdej przestrzeni znajdują się zmienne imię, humor, głód oraz zmęczenie
- W każdej przestrzeni znajdują się funkcje baw\_sie(czas), odpoczywaj(czas), jedz(ilosc)
- Funkcje baw\_sie(czas) poprawiają humor ale zwiększają głód oraz zmęczenie zwierzęcia (proporcjonalnie do czasu zabawy)
- Funkcja jedz(ilosc) zmniejsza głód zwierzęcia (w zależności od ilości jedzenia)
- Funkcja odpoczywaj(czas) zmniejsza zmęczenie zwierzęcia (proporcjonalnie do czasu)
- Ponieważ zabawa kota jest uciążliwa dla psa i vice-versa funkcja baw\_sie(czas) kota pogarsza humor psa i na odwrót.

W programie ustaw i wyświetl imię zwierzęcia oraz wypróbuj wszystkie zdefiniowane funkcje. Przynajmniej raz wyświetl stan wszystkich „parametrów zwierzęcia”.

## Referencje

W języku C dostępne są tylko zwykłe zmienne oraz wskaźniki do nich. W języku C++ wprowadzono referencje, które mają możliwości podobne do składników. Ich zaletą jest większa czytelność kodu – składnia pozwalająca na korzystanie z referencji nie różni się od składni zwykłych zmiennych.

```
int zmienna = 10; //zmienna i jej wartość początkowa
int &referencja_do_zmiennej = zmienna; //referencja musi być zainicjowana
//referencję można zainicjować tylko
//raz
cout << "zmienna: " << zmienna << endl; //wyświetli 10
referencja_do_zmiennej = 20; //modyfikacja zmiennej za pomocą
//referencji
cout << "zmienna: " << zmienna << endl; //wyświetli 20
cout << "referencja: " << referencja_do_zmiennej << endl; //wyświetli 20
```

Również korzystanie ze struktur/obiektów jest prostsze niż w przypadku korzystania ze wskaźników. Nie ma potrzeby stosowania operatorów \* czy ->.

```
Struktura struktura;
Struktura &referencja_do_struktury = struktura;
struktura.pierwsze = 1;
struktura.drugie = 2;
cout << "pierwsze: " << struktura.pierwsze << " drugie: "
    << struktura.drugie << endl;
referencja_do_struktury.pierwsze = 10;
referencja_do_struktury.drugie = 20;
cout << "pierwsze: " << struktura.pierwsze << " drugie: "
    << struktura.drugie << endl;
cout << "pierwsze: " << referencja_do_struktury.pierwsze << " drugie: "
    << referencja_do_struktury.drugie << endl;
```

przy czym Struktura jest zdefiniowana następująco:

```
struct Struktura
{
    int pierwsze;
    int drugie;
};
```

Referencje ułatwiają również przekazywanie danych do/z funkcji. Jak widać w przykładzie poniżej parametry do funkcji można przekazywać przez referencje. Odwoływania do pól struktury/obiektu są identyczne w przypadku referencji i zwykłych zmiennych (obiektów)

```
void funkcja(Struktura& parametr_referencja) //nie jest wykonywana kopia
//struktury = jest szybciej
```

```

{
    cout<< "początek funkcji" << endl;
    cout << "pierwsze: " << parametr_referencja.pierwsze << " drugie: "
        << parametr_referencja.drugie << endl; //dane przekazane z zewnątrz
    parametr_referencja.pierwsze=100;
    parametr_referencja.drugie=200;
    cout << "pierwsze: " << parametr_referencja.pierwsze << " drugie: "
        << parametr_referencja.drugie << endl; //zmodyfikowane dane wewnątrz
                                                //funkcji
    cout<< "koniec funkcji" << endl;
}

```

Ponieważ w przypadku przekazywania parametrów przez referencje nie jest wykonywana kopia danych – modyfikowane są oryginalne dane. Innymi słowy jest to sposób na przekazywanie danych z funkcji.

```

funkcja(struktura);
cout << "pierwsze: " << struktura.pierwsze << " drugie: "
    << struktura.drugie << endl; //zmodyfikowana struktura w głównym programie

```

Możliwe jest również tworzenie stałych referencji tzn. takich, za pomocą których nie można zmodyfikować wartości zmiennej. Ważne jest to, że samą zmienną można zmodyfikować.

```

int zmienna=100;
const int &referencja_do_zmiennej=zmienna;
zmienna=1000; //to zadziała
//referencja_do_zmiennej=2000; //na to kompilator nie pozwoli

```

Stałe referencje często znajdują zastosowanie jako argumenty funkcji. Przyspieszają przekazywanie danych a równocześnie dają pewność (tak na 99% ;-)) programiście korzystającemu z funkcji, że dane nie zostaną zmodyfikowane.

```

void funkcja2(const Struktura& parametr_referencja) //nie jest wykonywania
                                                    //kopia struktury = jest
                                                    //szybciej
{
    cout<< "początek funkcji" << endl;
    cout << "pierwsze: " << parametr_referencja.pierwsze << " drugie: "
        << parametr_referencja.drugie << endl; //dane przekazane z zewnątrz
    //tego nie można zrobić - parametr tylko do odczytu
    //parametr_referencja.pierwsze=100;
    //parametr_referencja.drugie=200;
    cout<< "koniec funkcji" << endl;
}

```

## Zadanie 5

Zmodyfikuj program z zadania szóstego tak aby korzystał z referencji.

- Zdefiniuj strukturę stan, która zawiera parametry zwierzęcia (przenieś do struktury informacje o humorze, głodzie oraz zmęczeniu zwierzęcia).
- Przerób funkcje tak aby miały dodatkowy parametr - referencje do struktur przechowujących stan zwierząt (tam gdzie to możliwe użyj stałych referencji).

## Zarządzanie pamięcią

W C++ wprowadzono nowe operatory new i delete do zarządzania pamięcią. Można również korzystać z funkcji malloc() i free() znanych z języka C jednak należy pamiętać, że nie wolno mieszać tych dwóch sposobów – tzn. pamięci zajętej przez malloc() nie można zwolnić za pomocą

delete a pamięci zajętej przez new zwolnić za pomocą free(). Należy również pamiętać, że błędem jest dwukrotne zwolnienie tej samej pamięci.

Poniżej przedstawiono przykłady alokacji i zwolnienia pamięci dla:

jednej zmiennej,

```
int *wskaznik_do_int = new int;
*wskaznik_do_int = 1;
cout << *wskaznik_do_int << endl;
delete wskaznik_do_int;
wskaznik_do_int=NULL; //bez tego byłby błąd podwójnego zwolnienia
delete wskaznik_do_int;
```

tablicy zmiennych,

```
int *tablica_int = new int[10];
for (int i = 0; i < 10; ++i)
    tablica_int[i] = i;
for (int i = 0; i < 10; ++i)
    cout << tablica_int[i] << endl;
delete[] tablica_int;
```

struktury/obiektu,

```
Struktura *wskaznik_do_struktury = new Struktura;
wskaznik_do_struktury->pierwsze = 10000;
(*wskaznik_do_struktury).drugie = 20000;
cout << "pierwsze: " << wskaznik_do_struktury->pierwsze << " drugie: "
    << (*wskaznik_do_struktury).drugie << endl;
delete wskaznik_do_struktury;
```

tablicy struktur/obiektów

```
Struktura *tablica_struktur = new Struktura[10];
for (int i = 0; i < 10; ++i)
{
    tablica_struktur[i].pierwsze = i;
    tablica_struktur[i].drugie = i + 1;
}
for (int i = 0; i < 10; ++i)
    cout << "pierwsze: " << tablica_struktur[i].pierwsze << " drugie: "
        << tablica_struktur[i].drugie << endl;
delete[] tablica_struktur;
```

## Zadanie 6

Zmodyfikuj program z zadania 5-go tak aby korzystał wyłącznie ze wskaźników.

- Zadbaj o właściwe zwolnienie pamięci przy zakończeniu programu
- Utwórz tablicę kotów i tablicę psów (tablice struktur opisujących stan zwierząt).
- Zmodyfikuj wszystkie funkcje aby zamiast referencji korzystały ze wskaźników

## Rzutowanie

W C++ wprowadzono nowe rodzaje rzutowania zmiennych. Do najważniejszych należą: `const_cast<>()`, `static_cast<>()`, `reinterpret_cast<>()` oraz `dynamic_cast<>()`, który zostanie omówiony później. Pierwszy z wymienionych rodzajów pozwala dodanie lub pozbycie się działania słowa kluczowego `const`. Rzutowanie `const_cast<>()` można stosować zarówno do referencji jak i

do wskaźników.

```
int liczba=1;
const int &stala_referencja=liczba;
cout<<"wartość początkowa: "<<stala_referencja<<endl;
//stala_referencja=2; //tak się nie da

const_cast<int &>(stala_referencja)=2; //ale kompilator można przekonać
cout<<"nowa wartość: "<<stala_referencja<<endl;

int &niestala_referencja=const_cast<int &>(stala_referencja);
niestala_referencja=3;
cout<<"nowsza wartość: "<<niestala_referencja<<endl;

const int& ponownie_stala_referencja=niestala_referencja;
//ponownie_stala_referencja=4;
```

Dwa pozostałe rodzaje rzutowania to `static_cast<>()` oraz `reinterpret_cast<>()`. Rzutowanie statyczne służy do rzutowania typów wbudowanych na inne typy wbudowane (z konwersją wartości) lub rzutowanie wskaźników do obiektów klasy bazowej na wskaźniki do obiektów klasy pochodnej (przy czym w trakcie działania programu nie jest wykonywane sprawdzenie poprawności tego rzutowania). `reinterpret_cast<>()` służy do rzutowania między wskaźnikami do liczb/obiektów typów niezwiązanych ze sobą. Możliwe jest też rzutowanie między liczbami a wskaźnikami.

```
float a=1;

cout<<"a: "<<a<<endl;
cout<<"static cast: "<<static_cast<int>(a)<<endl;
cout<<"reinterpret cast - adres a: "<<reinterpret_cast<unsigned int>(&a)<<endl;
```

Z rzutowaniem wiąże się jeszcze możliwość programowego sprawdzania typów zmiennych za pomocą operatora `typeid`.

```
int A,B;
float C;

cout<<typeid(A).name()<<" "<<typeid(B).name()<<" "<<typeid(C).name()<<endl;

if (typeid(A)==typeid(B))
    cout<<"A i B - te same typy"<<endl;
else
    cout<<"A i B - różne typy"<<endl;

if (typeid(A)==typeid(C))
    cout<<"A i C - te same typy"<<endl;
else
    cout<<"A i C - różne typy"<<endl;
```

## Zadanie 7

Za pomocą nowych typów rzutowania spowoduj żeby poniższy kod kompilował się, działał poprawnie i żeby wyniki obliczeń były dokładne. **Nie zmieniaj deklaracji/typów zmiennych oraz parametrów funkcji.**

```
#include <iostream>
using namespace std;

void funkcja(int &a)
{
```

```
    unsigned int adr=&a;
    cout<<"adres (dziesiętnie): "<<adr<<endl;
}

int main() {
    int i=10;
    const int &r=i;
    cout<<"adres (dziesiętnie): "<<&i<<endl;
    funkcja(r);
    int j=15;
    cout<<"i="<<i<<" j="<<j<<" a i/j="<<i/j<<endl;
    return 0;
}
```