Powtórka z C

Wykład 2

Sposób zapisu kodu

- W każdym programie musi być specjalna funkcja `main()`
- Instrukcje wykonywane w ramach tej funkcji zawarte są pomiędzy nawisami `{}`
- C++ zapis programu jest w tzw. formacie wolnym w przeciwieństwie do np. Fortran'a
  - Poza szczególnymi przypadkami łamanie linii może odbywać się w dowolnym miejscu
  - Dlatego każda instrukcja kończy się '`;`'
  - Białe znaki są prawie zawsze ignorowane

Pierwszy program

- Program na początku uruchamia funkcję main

```
#include <iostream>
using namespace std;
int main()
{
  cout << „Witam\n”;
  //(return 0;)
}
```

- Przykład cpp_2.01

Pierwszy program ...

- Przykład innego (złego) formatu zapisu kodu programu

```
#include <iostream>
using namespace std;main(){cout <<„Witam\n”;}
```

- Nie należy stosować, ponieważ
  - Kod staje się nieczytelny
  - Powoduje trudności w używaniu tzw. debugger'ów

Kompilacja i linkowanie

- Kod programu zostaje przetłumaczony na kod maszynowy w dwóch etapach
  - Kompilacja – pierwszy etap, tworzenie pików *.obj lub *.o (objects)
  - „Linkowanie" – łączenie plików obiektów z modułami bibliotecznymi
- Często proces „linkowania" jest niewidoczny dla programisty

Standardowe wejście/wyjście

- Potrzebny plik nagłówkowy
  - Umożliwia korzystanie z funkcji bibliotecznych
  - Dołączanie przez dyrektywę preprocesora
    ```
    #include <iostream.h>
    ```
  - ```
    #include <iostream>
    using namespace std;
    ```
    ```

Standardowe wejście/wyjście...

- Wypisywanie na ekran
  - `cout << „Tekst”;`
  - `cout << „\n”; // cout << endl;`
  - `cout a << b << endl;`
- Pobieranie danych z klawiatury
  - ```
    float a;
    cin >> a;
    ```
    ```

Komentarze

- Są zupełnie ignorowane przez kompilator
- Rodzaje
  - `/* ... */` - wiele linii, brak zagnieżdżeń
    - Czasami kompilator pozwala na zagnieżdżenia, ale to jest wbrew standardowi
  - `//` - do końca linii
- Komentarze są bardzo przydatne
  - W znakomity sposób ułatwiają zrozumienie kodu
  - Program Doxygen umożliwia automatyczne generowanie dokumentacji
    - Adres: www.doxygen.org

Nazwy

- Dowolny ciąg liter, cyfr oraz znaku '_'
- Nazwa nie może zaczynać się od cyfry
- Małe i duże litery są rozróżniane
- Nazwa nie może być identyczna z następującymi słowami kluczowymi języka C++:
  - `asm` - wstawia instrukcje asemblera
  - `auto` - deklaruje zmienną automatyczną
  - `bool` - deklaruje zmienną typu logiczną prawda/fałsz
  - `break` - przerywa działanie pętli
  - `case` - cześć instrukcji `switch`
  - `catch` - łapie wyjątki
  - `char` - deklaruje zmienną typu znak
  - `class` - deklaruje klasę
  - `const` - deklaruję zmienna typu stałego
  - `const_cast` - rzutowanie uzmieniające stałą
  - `continue` - wymusza ponowne wykonanie pętli
  - `default` - domyślna cześć instrukcji `switch`
  - `delete` - zwalnia pamięć

Słowa kluczowe ...

- **do** - część pętli **do/while**
- **double** - deklaruje zmienna typu rzeczywistego o podwójnej precyzji
- **dynamic_cast** - rzutowanie w czasie trwania programu
- **else** - część wyrażenie warunkowego **if**
- **enum** - definiuje typ wyliczeniowy
- **explicit** - w konstruktorze zabrania domyślnych konwersji
- **extern** - definiuje zmienną zewnętrzną
- **false** - wartość typu **bool** oznaczająca fałsz
- **float** - deklaruje zmienna typu rzeczywistego
- **for** - pętla
- **friend** - deklaruje przyjaźń
- **goto** - wykonuje skok
- **if** - instrukcja warunkowe
- **inline** - wstawia funkcję w linii
- **int** - deklaruje zmienną typu całkowitego
- **long** - deklaruje zmienną typu całkowitego o większym lub równym zakresie niż **int**
- **mutable** - uzmiennia stałą

Słowa kluczowe ...

- `namespace` - przestrzeń nazw
- `new` - alokuje pamięć
- `operator` - tworzy przeładowaną funkcję operatorową
- `private` - deklaruje prywatne składniki i metody klasy
- `protected` - deklaruje zabezpieczone składniki i metody klasy
- `public` - deklaruje publiczne składniki i metody klasy
- `register` - optymalizuje dostęp do zmiennej ze względu na szybkość
- `reinterpret_cast` - zmienia typ zmiennej
- `return` - zwraca wartość z funkcji
- `short` - deklaruje zmienną typu całkowitego o mniejszym lub równym zakresie niż `int`
- `signed` – określa, że zmienna jest ze znakiem
- `sizeof` - zawraca rozmiar typu lub zmiennej
- `static` - tworzy zmienną która istnieje przez całe wykonywanie się programu
- `static_cast` - operator rzutowanie
- `struct` - tworzy strukturę
- `switch` - wykonuje kod zależny od danej zmiennej

Słowa kluczowe ...

- `template` - tworzy szablon
- `this` - wskaźnik do obecnie używanego obiektu
- `throw` - wyrzuca wyjątek
- `true` - wartość typu `bool` oznaczająca prawdę
- `try` - wykonuje kod który może wyrzucić wyjątek
- `typedef` - tworzy synonim do istniejącego typu
- `typeid` - opisuje typ obiektu
- `typename` - w szablonach oznacza że następujący po nim symbol reprezentuje typ (synonim `class`)
- `union` - tworzy unię
- `unsigned` - deklaruje zmienną bez znaku
- `using` - używa przestrzeni nazw
- `virtual` - tworzy funkcję wirtualną
- `void` - deklaruje zmienną z nieprzypisanym typem
- `volatile` - ostrzega kompilator że zmienna może zostać zmodyfikowana nieoczekiwanie
- `wchar_t` - deklaruje "szeroką" zmienną znakową
- `while` - pętla z wyrażeniem warunkowym

Zmienne

- W języku C++ każda nazwa musi być zadeklarowana przed jej użyciem
- Deklaracja – mówi tylko że dana nazwa będzie oznaczała obiekt typu ...
  - `extern int x;`
- Definicja – rezerwuję pamięć dla obiektu typu ...
  - `int x = 0;`
- Definicja jest równocześnie deklaracją, ale nie odwrotnie

Typy

- Dwa podziały typów
  - Pierwszy
    - Typy fundamentalne (podstawowe)
    - Typy pochodne
  - Drugi
    - Typy wbudowane
    - Typy zdefiniowane przez użytkownika

Typy fundamentalne

- Reprezentujące liczby całkowite
  - `short int (short)`
  - `int`
  - `long int (long)`
- Reprezentujące znaki alfanumeryczne
  - `char`
  - `wchar_t`
- Modyfikatory (dla powyższych)
  - `signed` - liczba ujemna i dodatnia
  - `unsigned` - tylko liczba dodatnia
- Reprezentujące liczby zmiennoprzecinkowe
  - `float`
  - `double`
  - `long double`

Stałe liczbowe

- Całkowite
  - Dziesiątkowe
    - 13, -55, 0, 1000, ...
  - Ósemkowe
    - 010 = 8
    - 013 = 11
    - 091 – błąd
  - Szesnastkowe
    - 0x10 = 16
    - 0xFF = 255
  - Przykład cpp_2.2
- Zmiennoprzecinkowe
  - 0.0, 3.14159, -1000.0, 8e2, 13.3e-13
  - Przykład cpp_2.3

Stałe znakowe

- Reprezentują znaki alfanumeryczne
  - `char a = '7';`
  - `char b = 'a';`
- Różne sposoby kodowania np. ASCII
- Znaki specjalne
- `'\b'` – backspace, `'\f'` – nowa strona, `'\n'` – nowa linia, `'\r'` – powrót karetki, `'\t'` – tabulator poziomy, `'\v'` – tabulator pionowy, `'\a'` – sygnał dźwiękowy

Stałe znakowe ...

- Znaki których nie da się zapisać bezpośrednio w apostrofach
  - '\\' - backslash
  - '\'' - apostrof
  - '\"' - cudzysłów
  - '\0' - NULL
  - '\?' - pytajnik
- Stałe tekstowe
  - Znaki ujęte pomiędzy cudzysłowami ""

Typy pochodne

- Tablice - `[]`
- Wskaźniki - `*`
- Funkcje - `()`
- Referencje - `&`
- Typ `void`
  - funkcja nic nie zwraca `void licz();`
  - wskaźnik wskazuje na nieznany typ
    `void *p;`

Typ wyliczeniowy

- **enum**
  - Osobny typ całkowity
  - Bardzo przydatny jeśli potrzebujemy przechowywać jakiś rodzaj informacji
  - ```
    enum status {
    start = 0,
    stop = 1,
    error = -1 };
    ```
  - Wielką zaletą jest, że kompilator „pilnuje" czy wysyłamy poprawny argument
    ```

Zakres obowiązywania nazw

- Lokalny
  - Ograniczony nawiasami klamrowymi {}

```
main()
{
     {       //początek obszaru lokalnego
             int x;
             ...
     }       //koniec obszaru lokalnego
     ...
}
```

- Blok funkcji
  - Ograniczony do danej funkcji
  - Z powyższego wynika iż nie można używać `goto` do skoków między funkcjami

Zakres obowiązywania nazw...

- Obszar pliku
  - Jeśli zadeklarujemy na zewnątrz jakiegokolwiek bloku – nazwa staje się globalna (ale w danym pliku)

```
double f; // nazwa globalna
main()
{
   ...
}
```

- Obszar klasy

Zasłanianie nazw

- Możliwe jest zdefiniowanie nazwy lokalnej, która zasłoni zmienną globalną

```
int k = 13;
int main()
{
   cout << "k = " << k << endl;
   {
            int k = 1313;
            cout << "k (lokalne) = " << k << endl;
            cout << "k (globalne z bloku lokalnego) = "
   << ::k << endl;
   }
   cout << "k (po bloku lokalnym) = " << k << endl;
}
```

- Przykład cpp_2.4

Inne modyfikatory

- **`const`**
  - Obiekty z tym modyfikatorem nie mogą być zmieniane w programie

```
const float pi = 3.14159; //inicjalizacja
pi = 10;    //błąd – próba przypisania
```

  - Obiekty typu **`const`** można inicjalizować, ale nie można do nich nic przypisać
    - Czy aby na pewno?
- **`register`**
  - Umieszczanie zmiennej w rejestrze – szybszy dostęp
  - Nie można uzyskać adresu zmiennej jeśli chcemy, aby nie została przeniesiona do zwykłej pamięci

Inne modyfikatory…

- **`volatile`**
  - Oznacza obiekt ulotny – może się zmienić w sposób niezauważalny dla kompilatora
  - Ma zawsze zostać odczytany z pamięci
  - Nie używa buforowania (cache-u)
- **`mutable`**
  - Umożliwia uzmiennienie stałej, przydatne podczas tworzenia klas

Inne modyfikatory…

- **`auto`** i **`static`**
  - Obiekty globalne są typu **`static`** (statyczne), a lokalne typu **`auto`** (automatyczne)
    - Do zmiennych automatycznych należy najpierw coś zapisać, a dopiero potem czytać (nie są zerowane)
    - Zmienne statyczne są przed uruchomieniem programu zerowane
    - Obiekty lokalne można deklarować jako statyczne (np. zapamiętanie wartości zmienne przy kolejnym wywołaniu funkcji)
- Przykład cpp_2.5

Instrukcja typedef

- Pozwala na nadanie dodatkowej nazwy już istniejącemu typowi
  - ```
    typedef float press;
    press a, b ,c;
    ```
  - Przydatne np. w sytuacji kiedy nagle typ `float` powinien ulec zmianie na inny – nie trzeba przepisywać programu
  - Nie tworzymy nowego typu, ale tworzymy synonim do już istniejącego
    ```

Operatory

- Arytmetyczne
  - Dwuargumentowe
    - `+` dodawanie
    - `-` odejmowanie
    - `*` mnożenie
    - `/` dzielenie
    - `%` modulo
    - `=` przypisanie
  - Jednoargumentowe
    - `++` inkrementacja
      - Preinkrementacja `++a;`
      - Postinkremetacja `a++;`
    - `--` dekrementacja
      - Predekrementacja `--a;`
      - Postdekrementacja `a--;`
- Przykład cpp_2.6

Operatory...

- Logiczne
  - relacji
    - `>` większy
    - `<` mniejszy
    - `<=` większy lub równy
    - `>=` mniejszy lub równy
    - `==` równy
  - `||` - suma logiczna (alternatywa)
  - `&&` - iloczyn logiczny (koniunkcja)
  - `!` - zaprzeczenie (negacja)
- Przykład cpp_2.7

Operatory...

- Bitowe
  - << przesunięcie w lewo
  - >> przesunięcie w prawo
  - | suma bitowa (OR)
  - & iloczyn bitowy (AND)
  - ^ różnica symetryczna bitów (XOR)
  - ~ negacja bitów (NOT)
- Przykład cpp_2.8

Operatory...

- Przypisania (pozostałe)
  - +=
  - -=
  - *=
  - /=
  - %=
  - >>=
  - <<=
  - &=
  - |=
  - ^=

Operatory...

- `sizeof(nazwa)`
  - Podaje wielkość typów, także zdefiniowanych przez programistę
- Rzutowania znane z C (nie zalecane)
  - `(nazwa_typu)obiekt`
- Przecinek ','
  - Stosowany kilku wyrażeń stojących obok siebie traktowanych jako jedno o wartości wyrażenia będącego najbardziej z prawej
- Wyrażenie warunkowe
  - `(x > y) ? 1 : 0`
- Przykład cpp_2.9

Priorytety operatorów

Prio.	Symbol	Łącz.	Prio.	Symbol	Łącz.
17	`::` - zakres `::` - nazwa globalna	L P	8	`&`	L
16	`. -> [] ()`		7	`^`	L
15	`sizeof ++ -- ~ ! - + & * new delete ()`- rzut.	P	6	`|`	L
14	`.* ->*`	L	5	`&&`	L
13	`* / %`	L	4	`||`	L
12	`+ -`	L	3	`?:`	L
11	`<< >>`	L	2	`= *= /= %= += -= <<= >>= &= |= ^=`	P
10	`< <= > >=`	L	1	`,`	L
9	`== !=`	L			

Łączność operatorów

- Łączność lewostronna (L)
  - a * b *c * d
  - (((a * b) *c) * d)
- Łączność prawostronna (P)
  - a = b = c = d
  - (a = (b = (c = d)))

Instrukcja warunkowa `if`

- ```
  if (wyr1)
     inst1;
  else if (wyr2)
     inst2;
  else
     inst3;
  ```
- Przykład cpp_2.10
  ```

Pętla `while` i `do while` i `for`

- `while(wyr)`
  `    inst1;`
  - Przykład cpp_2.11
- `do`
  `    inst1;`
  `while(wyr);`
  - Przykład cpp_2.12
- `for(inst_init; wyr; krok)`
  `    inst2;`
- `inst_init` i `krok` mogą być instrukcjami wielokrotnymi oddzielonymi przecinkami
  - Przykład cpp_2.13

Instrukcja switch

```
switch(wyr)
{
    case wart1:
         inst1;
    case wart2:
         inst2; break;
    default:
         inst2; break;
}
```

- Wyrażenie `wyr` może być tylko `int`
- Przykład cpp_2.14

Instrukcje `break`, `continue`, `goto`

- **`break`** - przerywa wykonywanie pętli
- **`continue`** - przerywa dany obieg pętli
  - Przykład cpp_2.15
- **`goto etykieta`** - instrukcja skoku, której dobry programista nie używa (ewentualnie można użyć do opuszczenie wielokrotnie zagnieżdżonej pętli)
  - Nie ma przykładu bo dobrze zorganizowany kod jej nie potrzebuje