Podstawy Programowania

Wykład nr 2: Pierwsze programy.

dr hab. inż. Dariusz Dereniowski

Katedra Algorytmów i Modelowania Systemów
Wydział ETI, Politechnika Gdańska

Struktura programu

- Kod programu znajduje się w jednym bądź kilku plikach.
- Program składa się z dyrektyw preprocesora, definicji struktur, funkcji oraz zmiennych.
- Funkcje zbudowane są z instrukcji.
- Instrukcje składają się ze słów kluczowych, operatorów, nazw zmiennych oraz znaków grupujących i separujących (nawiasy, przecinki, średniki).
- Program zaczyna działanie od wykonania funkcji main.
- Pozostałe funkcje mogą być napisane przez programistę lub pochodzić z bibliotek.
- Przed uruchomieniem, program należy skompilować i scalić z funkcjami bibliotecznymi.

Drugi program

```
/* dyrektywy preprocesora: włączenie plików nagłówkowych */
#include <iostream>
#include <math.h>
using namespace std;

/* definicja funkcji main, zwracającej wartość całkowitą */
int main () {
        /* deklaracja zmiennych */
        double a, b;

        cout << "Podaj długości boków prostokąta\n";
        cin >> a;
        cin >> b;
        /* instrukcja przypisania, operacje arytmetyczne, wywołanie funkcji
             bibliotecznej */
        double c = sqrt(a*a + b*b);
        cout << "Długość przekątnej prostokąta wynosi: ";
        cout << c << '\n';
        return 0;  /* wartość zwracana typu całkowitego */
}
```

Jednostki leksykalne

- Jednostki leksykalne to podstawowe “cegiełki”, z których zbudowany jest program.
- Jednostki leksykalne oddzielone są od siebie separatorami (spacja, przecinek, średnik, znak końca linii).
- Przykłady z poprzedniego programu:
  - int, double, return — słowa kluczowe
  - main, sqrt — identyfikatory (nazwy funkcji)
  - a, b, c — identyfikatory (nazwy zmiennych)
  - 0 — stały literał liczbowy
  - ”Podaj długości boków prostokąta-– stały literał znakowy
  - komentarze są usuwane z programu przez preprocesor
  - dyrektywy są zastępowane odpowiednimi definicjami przez preprocesor

Identyfikatory

- Identyfikatory opisują nazwy zmiennych, funkcji, stałych
- Identyfikator jest sekwencją znaków alfanumerycznych, tj. liter i cyfr oraz znaków podkreślenia ('_')
- Pierwszy znak nie może być cyfrą
- Rozróżniane są małe oraz duże litery
- Długość identyfikatora jest nieograniczona, ale znaczące są tylko początkowe znaki identyfikatora (minimum 31 dla identyfikatorów zewnętrznych, minimum 63 w obrębie kodu)
- Należy unikać nazw zaczynających się od '_' (wykorzystywane w bibliotekach, zwłaszcza przy scalaniu z fragmentami kodu w innych językach)

Słowa kluczowe

- 32 słowa kluczowe w standardzie C89 (auto, break, case, char, const, continue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while)
- 5 kolejnych dodanych w standardzie C99 (_Bool, _Complex, _Imaginary, inline, restrict)
- 7 kolejnych w standardzie C11 (_Alignas, _Alignof, _Atomic, _Generic, _Noreturn, _Static_assert, _Thread_local)
- nie mogą być używane jako identyfikatory
- podstawowe typy danych: char, int, float, double, void
- złożone typy danych (użytkownika): struct, union
- aliasy typów danych: typedef, enum
- operator rozmiaru typu danych: sizeof
- modyfikatory typów numerycznych (domyślnie dla int): long, short, signed, unsigned
- modyfikatory zasięgu/widoczności zmiennych i funkcji: const, static, volatile, extern, register, auto
- strukturalne instrukcje sterujące: for, while, do, switch, case, default, if, else
- instrukcje sterujące skoków/powrotów: break, continue, return, goto

Deklaracja zmiennej

- Zmienne w języku C określają pewien obszar w pamięci komputera.
- Identyfikator zmiennej to nazwa, poprzez którą programista może odwoływać się do wartości zmiennej.
- W obrębie danego bloku instrukcji (pomiędzy klamrami { oraz } oraz wewnątrz niektórych instrukcji) identyfikator powinien być unikatowy.
- Wartości zmiennej mogą być modyfikowane w trakcie działania programu.
- Przykłady deklaracji zmiennych:

```
int i;
char i,j;
int i=0;
char c = 'f';
int i,j=0;
```

Instrukcja przypisania

```
zmienna = wyrażenie;
```

- zmienna $\Rightarrow$ l-wartość
- Jeśli typy zmiennej oraz wyrażenia są różne, to następuje automatyczna konwersja do typu zmiennej.
- różne warianty dla zmiennych liczbowych:

```
zmienna ○= wyrażenie;
```

... gdzie ○ zastępujemy operatorem, np.:

```
a *= 5;
```

- Przypisanie postaci a ○= b jest interpretowane przez kompilator jako a = a ○ b. W miejsce ○ możemy wstawić: + - * / % << >> & | ^

Operatory arytmetyczne

- addytywne. Konieczna zgodność typów: oba operandy (a oraz b) typów arytmetycznych, lub jeden wskaźnikowy a drugi całkowity.
  - dodawanie: a+b
  - odejmowanie: a-b
- multiplikatywne
  - mnożenie: a*b (operandy arytmetyczne)
  - dzielenie: a/b (operandy arytmetyczne) W przypadku operatorów całkowitych następuje zaokrąglenie w dół.
  - reszta z dzielenia liczby a przez liczbę b: a%b (operandy całkowite)
- inkrementacji/dekrementacji (zwiększenie/zmniejszenie o 1)
  - przedrostkowe: --a lub ++a. (Zmiana wartości a następuje przed obliczeniem wartości wyrażenia.)
  - przyrostkowe: a-- lub a++ (Zmiana wartości a następuje po obliczeniu wartości wyrażenia.)

Operatory arytmetyczne – przykład

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

/* Program ilustrujący operatory arytmetyczne */
int main() {
  int a = 10, b = 7;

  b = (a++) % b;
  cout << "a=" << a << ",b=" << b << endl;     /*a=11,b=3*/
  cout << "a=" << a << ",b=" << b++ << endl;   /*a=11,b=3*/
  cout << "a=" << --a << ",b=" << b << endl;   /*a=10,b=4*/
  a = 7/10;
  b = 7/2;
  cout << "a=" << a << ",b=" << b << endl;   /*a=0,b=3*/

  return 0;
}
```

Instrukcje

- Instrukcja to wyrażenie zakończone średnikiem (który pomijamy w przypadku użycia grupowania).
- Instrukcje można grupować używając nawiasów klamrowych.
- Podstawowe typy instrukcji:
  - przypisania (=)
  - warunkowe (if, if-else)
  - wyboru (switch)
  - pętle (while, do-while, for)
  - skoki (goto, continue, break)

Operatory relacyjne

Operatory relacyjne:

- a <= b (a mniejsze lub równe b)
- a >= b (a większe lub równe b)
- a < b (a mniejsze niż b)
- a > b (a większe niż b)
- a == b (a równe b)
- a != b (a różne od b)

Oba operandy muszą być typów arytmetycznych lub muszą być wskaźnikami zgodnych typów.

Instrukcja warunkowa

```
if (wyrażenie)
   instrukcja1;
```

lub

```
if (wyrażenie)
   instrukcja1;
else
   instrukcja2;
```

1. Obliczana jest wartość wyrażenia w nawiasach.
2. Jeśli wartość ta jest niezerowa, to wykonywana jest instrukcja1.
3. Jeśli wartość ta jest równa zero, to:
   - W przypadku po lewej stronie: żadne dodatkowe instrucje nie są wykonywane.
   - W przypadku po prawej stronie: wykonywana jest instrukcja2.

Instrukcja warunkowa – przykład

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

/* Program ilustrujący użycie instrukcji warunkowej */
int main() {
    int liczba1, liczba2;

    cin >> liczba1;
    cin >> liczba2;

    if ( liczba2 != liczba1 )
        cout << "liczba1 jest różna od liczba2\n";
    if ( liczba2 == liczba1 )
        cout << "liczba1 == liczba2\n";
    return 0;
}
```

Instrukcja warunkowa – przykład

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

/* Program ilustrujący użycie instrukcji warunkowej */
int main() {
    int liczba1, liczba2;

    cin >> liczba1;
    cin >> liczba2;

    if ( liczba2 != liczba1 )
        cout << "liczba1 jest różna od liczba2\n";
    else
        cout << "liczba1 == liczba2\n";
    return 0;
}
```

Instrukcja warunkowa – przykład

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

/* Program ilustrujący użycie instrukcji warunkowej */
int main() {
    int liczba1, liczba2;

    cin >> liczba1;
    cin >> liczba2;

    if ( liczba2 - liczba1 )
        cout << "liczba1 jest różna od liczba2\n";
    else
        cout << "liczba1 == liczba2\n";
    return 0;
}
```

Zagnieżdżona instrukcja warunkowa

```
if (wyrażenie1)
   instrukcja1;
else if (wyrażenie2)
   instrukcja2;
else if (wyrażenie3)
  instrukcja3;
else if (wyrażenie4)
  instrukcja4;
else
  instrukcja5;
```

Uwaga! Porównaj:

```
if (wyrażenie1)
{
   if (wyrażenie2) instrukcja2;
}
else instrukcja5;
```

oraz

```
if (wyrażenie1)
   if (wyrażenie2) instrukcja2;
else instrukcja5;
```

Instrukcja for

```
for ( ini; wyrażenie; next )
  instrukcja;
```

1. Relizacja instrukcji for rozpoczyna się od wykonania instrukcji ini.
2. Obliczana jest wartość wyrażenia.
3. Jeśli wartość ta jest zerowa, to następuje zakończenie wykonywania instrukcji for.
4. Jeśli wartość ta jest niezerowa, to wykonywana jest instrukcja (treść pętli).
5. Wykonywana (na zakończenie bieżącego obiegu pętli) jest instrukcja next i następuje powrót do pkt.2.

- Część ini lub next może się składać z kilku instrukcji – oddzielamy je przecinkami.
- Każdą z części ini, wyrażenie, next można pominąć. Należy jednak pozostawić oddzielające średniki.
- Jeśli wyrażenie jest puste, to jest ono traktowane jakby było prawdziwe.
- Użycie instrukcji break wewnątrz pętli powoduje zakończenie wykonywania pętli.

Instrukcja for – przykład

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

/* wypisanie liczb podzielnych przez 3 z zakresu [1,n],
   gdzie n jest wartością podaną przez użytkownika.*/
int main() {
    int n, i;

    cin >> n;
    for ( i=3; i <= n; i+=1 )
       if ( i % 3 == 0 )
          cout << i << endl;
    return 0;
}
```