Programowanie strukturalne - Wykład 2

Programowanie strukturalne

Programowanie strukturalne to paradygmat programowania, który skupia się na tworzeniu programów poprzez zastosowanie jedynie określonych struktur programistycznych, takich jak sekwencje, pętle i instrukcje warunkowe.

Celem programowania strukturalnego jest uproszczenie procesu tworzenia programów poprzez ograniczenie złożoności i zapewnienie łatwej zrozumiałości kodu. W tym paradygmacie programowania stosuje się także podejście top-down, które polega na dzieleniu programu na mniejsze, łatwiejsze do zrozumienia fragmenty, co ułatwia zarówno sam proces programowania, jak i późniejsze modyfikacje kodu.

Uznaje się, że program nie ma postaci strukturalnej, jeżeli występuje w nim:

- instrukcja skoku (goto) na zewnątrz pętli
- instrukcja skoku (goto) do środka instrukcji warunkowej
- instrukcja skoku (goto) do środka pętli
- instrukcja skoku (goto) do innej części programu

Sprawdzenie wersji języka C

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    printf("%d",__STDC_VERSION__);
    return 0;
}
```

Wbudowane typy danych

- https://en.wikipedia.org/wiki/C_data_types
- https://devdocs.io/c/language/arithmetic_types

Deklaracja zmiennej

```
typ nazwa_zmiennej;
```

Przykład:

```
int a;
```

Przykład z nadaniem wartości:

```
int a=7;
```

Instrukcje wyjścia/wejścia

- Szczegółowo będą później

`print` - wyjście

https://en.cppreference.com/w/c/io/fprintf

`scanf` - wejście

https://en.cppreference.com/w/c/io/fscanf

Formaty

Na razie do samodzielnej analizy:

https://gist.github.com/pjastr/100599c74e34cd814f671daccd0668b0

https://gist.github.com/pjastr/37e259e243a7b9d26e9c45d528c84c02

https://gist.github.com/pjastr/8c059084a44781c9407ee6ab0a3c5ef3

https://gist.github.com/pjastr/1285dbb398d7cfed9d372b7ed292d599

https://gist.github.com/pjastr/f59613feebb3900634cbe41658ae2be7

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a =0;
    scanf("%d",&a);
    printf("%d",a);
    return 0;
}
```

https://cpp0x.pl/dokumentacja/tekst-sformatowany-printf/736

Różnica między `%d` a `%i`?

Dotyczy tylko `scanf` przy `%i` mamy warianty:

- `12` - system dziesiętny
- `012` - system ósemkowy
- `0x12` - system szesnastkowy

Prezentacja debuggera

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a = 5;
    a= -2;
    a=7;
    return 0;
}
```

Zasięg zmiennej

- globalne - zadeklarowane poza `main`, dostępne dla wszystkich funkcji.
- lokalne - zadeklarowane w innym miejscu

```c
#include <stdio.h>

int a,b;

void func1 ()
{
    a=3;
}

int main ()
{
    b=3;
    a=2;
    return 0;
}
```

```c
int a=1; /* zmienna globalna */

int main()
{
    int a=2;          /* to już zmienna lokalna */
    printf("%d", a); /* wypisze 2 */
}
```

Czas życia

Czas życia to czas od momentu przydzielenia dla zmiennej miejsca w pamięci (stworzenie obiektu) do momentu zwolnienia miejsca w pamięci (likwidacja obiektu).

```
 1  int main()
 2  {
 3   int a = 10;
 4   {          /* otwarcie lokalnego bloku */
 5       int b = 10;
 6       printf("%d %d", a, b);
 7   }          /* zamknięcie lokalnego bloku */
 8
 9      printf("%d %d", a, b);  /* BŁĄD: b juz nie istnieje */
10  }
```

Stałe

Stała, różni się od zmiennej tylko tym, że nie można jej przypisać innej wartości w trakcie działania programu. Wartość stałej ustala się w kodzie programu i nigdy ona nie ulega zmianie.

```
1  const typ nazwa_stałej=wartość;
```

#define

`#define` - dyrektywa preprocessora.

Linia pozwalająca zdefiniować stałą, funkcję, słowo kluczowe lub makro, które będzie potem podmienione w kodzie programu na odpowiednią wartość lub może zostać użyte w instrukcjach warunkowych dla preprocesora.

```
1 #define NAZWA_STALEJ WARTOSC
```

```
1 #define NAZWA_STALEJ
```

```c
#include <stdio.h>

#define SIX 1+5
#define NINE 8+1

int main(void)
{
    printf("%d * %d = %d\n", SIX, NINE, SIX * NINE);
    return 0;

}
```

Operator przypisania

Operator przypisania (“=”), jak sama nazwa wskazuje, przypisuje wartość prawego argumentu lewemu.

```
1 int a = 3, b;
2 b = a;
3 printf("%d\n", b); /* wypisze 3 */
```

C umożliwia też skrócony zapis postaci a #= b;, gdzie # jest jednym z operatorów: +, -, *, /, %, &, |, ^, << lub >>. Ogólnie rzecz ujmując zapis a #= b; jest równoważny zapisowi a = a # b.

```
1  int a = 1;
2  a += 5;      /* to samo, co a = a + 5;       */
3  a /= a + 2; /* to samo, co a = a / (a + 2); */
4  a %= 2;      /* to samo, co a = a % 2;       */
```

Operatory arytmetyczne dwuargumentowe

- dodawanie (“+”),
- odejmowanie (“-”),
- mnożenie (“*”),
- dzielenie (“/”),
- reszta z dzielenia (“%”) określona tylko dla liczb całkowitych (tzw. dzielenie modulo).

Jednoargumentowe operatory arytmetyczne

- pre-inkrementacja (`++i`),
- post-inkrementacja (`i++`),
- pre-dekrementacja (`--i`),
- post-dekrementacja (`i--`).

```
1  int a, b, c;
2  a = 3;
3  b = a--; /* po operacji b=3 a=2 */
4  c = --b; /* po operacji b=2 c=2 */
```

Operator rozmiaru `sizeof()`

Zwraca rozmiar obiektu podany w wybranej jednostce miary, np. bajtach lub słowach maszynowych.

```
1 printf ("%d",sizeof(int));
```

```
1 4
```

Operatory bitowe

- negacja bitowa (NOT)(“~”),
- koniunkcja bitowa (AND)(“&”),
- alternatywa bitowa (OR)(“|”) i
- alternatywa rozłączna (XOR) (“^”)

Są zdefiniowane dla liczb całkowitych, działają na bitach i mogą być szybsze niż zwykłe operacje.

https://pl.wikibooks.org/wiki/C/Operatory#Operatory_bitowe

Operatory porównania

- równe (“==”),
- różne (“!=”),
- mniejsze (“<”),
- większe (“>”),
- mniejsze lub równe (“<=”)
- większe lub równe (“>=”)

Uwaga: porównywanie floatów nie za pomocą ==

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    float a=1.1;
    float b=3.3;
    printf("%d",b==3*a);
    return 0;
}
```

Operatory logiczne

- negacja (zaprzeczenie): “!”
- koniunkcja (“i”): “&&”
- alternatywa (“lub”): “||”

Operator wyrażenia warunkowego

```
1 a ? b : c
```

Najpierw oceniana jest wartość logiczna wyrażenia a; jeśli jest ono prawdziwe, to zwracana jest wartość b, jeśli natomiast wyrażenie a jest nieprawdziwe, zwracana jest wartość c.

Kolejność operatorów

https://en.cppreference.com/w/c/language/operator_precedence

Instrukcje sterujące

Instrukcje warunkowe

```
1  if (wyrażenie) {
2    /* blok wykonany, jeśli wyrażenie jest prawdziwe */
3  }
```

```
1   if (wyrażenie) {
2    /* blok wykonany, jeśli wyrażenie jest prawdziwe */
3  } else {
4    /* blok wykonany, jeśli wyrażenie jest nieprawdziwe */
5  }
```

```
switch (wyrażenie) {
case wartość1:
 break;
case wartość2:
 break;
/* ... */
default:
 break;
}
```

Przykłady:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a =5;
    if (a>0)
    {
        printf("Liczba dodania\n");
    }
    else
    {
        printf("Liczba nie jest dodania\n");
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a =-1;
    if (a)
    {
        printf("Prawda\n");
    }
    else
    {
        printf("Fałsz\n");
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int a =-1;
    if (a)
        printf("a");
    printf("bc");
    return 0;
}
```

Pętle

for

```
for (wyrażenie1; wyrażenie2; wyrażenie3) {
   /* instrukcje do wykonania w pętli */
 }
```

do ...while

```
1  do {
2    /* instrukcje do wykonania w pętli */
3  } while (warunek);
```

while

```
1  while (warunek) {
2     /* instrukcje do wykonania w pętli */
3   }
```

Przykłady pętli

Warunek początkowy może lub nie musi mieć deklarację zmiennej interującej (od C99).

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    for(int i=1;i<5;i++)
    {
        printf("%d\n",i);
    }
    return 0;
}
```

Przed C99

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    for(i=1;i<5;i++)
    {
        printf("%d\n",i);
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    for(int i=0;i<5;i++)
    {
        printf("%d\n",i);
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    for(int i=5;i>0;i--)
    {
        printf("%d\n",i);
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    for(int i=1;i*i<100;i+=2)
    {
        printf("%d\n",i);
    }
    return 0;
}
```

Nieskończona pętla

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    for(;;)
    {
        printf("endless loop!");
    }
    return 0;
}
```

Wnętrze jest opcjonalne:

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    for(int n = 0; n < 10; ++n, printf("%d\n", n))
    ; // null statement
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i =0;
    while (i<5)
    {
        printf("%d\n",i);
        i++;
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i =0;
    while (i<5)
    {
        printf("%d\n",i);
        if (i>2)
        {
            break;
        }
        i++;
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i =0;
    while (i<5)
    {
        i++;
        if (i==2)
        {
            continue;
        }
        printf("%d\n",i);
    }
    return 0;
}
```

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i =0;
    do
    {
        printf("%d\n",i);
        i++;
    }
    while (i<-2);
    return 0;
}
```

Bibliografia

- Wojciech Sobieski, Języki Programowania, http://pracownicy.uwm.edu.pl/wojsob/pliki/publikacje/jp-01.pdf, dostęp online 20.02.2020.
- https://pl.wikipedia.org/wiki/C_(j%C4%99zyk_programowania), dostęp online 20.02.2020.
- https://pl.wikibooks.org/wiki/C/Zmienne#Deklaracja_zmiennych, dostęp online 20.02.2020.
- https://pl.wikibooks.org/wiki/C/Operatory, dostęp online 20.02.2020.
- https://pl.wikibooks.org/wiki/C/Operatory#Operatory_bitowe, dostęp online 10.03.2020.

- https://pl.wikibooks.org/wiki/C/Funkcje, dostęp online 10.03.2020.
- https://pl.wikipedia.org/wiki/Rekurencja, dostęp online 10.03.2020.
- https://pl.wikibooks.org/wiki/C/Instrukcje_steruj%C4%85ce, dostęp online 10.03.2020.