Pierwsze kroki w C

Najprostszy użyteczny program w C

```
int main() {
  return 0;
}
```

kompilacja:

gcc true.c -o true

lub

make true

Pytanie:

Jak wypisać w baszu wynik wykonania programu?

```
./program ; echo $?
```

```
if ./a ; then echo ok; else echo no; fi
```

Struktura programu w C

program to lista:

- deklaracji (zmiennej albo funkcji)
- definicji funkcji

Program główny to funkcja o nazwie `main()`

W programie mogą się znaleźć też makra preprocesora (zaczynają się one od # np: **`#include , #declare`**)

Deklaracja zmiennej

<typ> <nazwa1> [= <wartość początkowa>] [, <nazwa2> ...] ;

np:

```
int a,c,d=4;
long a=3L;
char literka='z', znak='\n';
double wys=.1, szer=0., dlugosc=3.0;
long double duza_wys=0.1e-23L;
float masa=1.7e2f, masa2=1.7e3F, m3=0;
char * napis = "Ala ma kota";
```

Typy proste

```
unsigned, signed
char                                    //1
int, short int, long int, long, long
  long//4, 2, 4, 4,8
float, dluble, long dluble   //sprawdz
```

sizeof(<typ>**)**

sizeof <wyrażenie>

Deklaracje funkcji (prototyp)

Podobnie jak deklaracja zmiennej

```
int a(long int);
char b(), c(int);
void m(int a, int b);
void mmx(void);
```

Definicja funkcji

<typ> <nazwa>(<lista parametrów >)

{

[<deklaracje>]

[<instrukcje>]

}

Przykład funkcji

```
int f(int a, long dd) {
  int c = 3, b;
  b=dd % c;   // modulo
  return a+(c/a)-b;
}
int c( int a) {
  return f(a, 30);
}
```

Funkcje

- Zwracanie wyniku
  - **`return ;`**
  - **`return`** <wyrażenie> **`;`**
- Parametry funkcji przekazywane są zawsze przez wartość
- Definicje funkcji nie mogą być zagnieżdżone

Operatory

- `() [] -> .`
- `! ~ ++ -- - * (typ) sizeof` `//p`
- `* / %`
- `+ -`
- `<< >>`
- `< <= > >=`
- `== !=`
- `&`
- `^`
- `|`
- `&&`
- `||`
- `?:` `//p`
- `= += -= *= /= %= &= ^= |= <<= >>=` `//p`
- `,`

Uwaga

`&& , || , !` - operacje logiczne na

`0` – fałsz

`...-2,-1,1, 2, ...` – prawda

wyniki tylko `0, 1`

`& , | , ~ , ^, <<, >>` - operacje bitowe
and,or,not,xor, shl, shr

- tylko dla całkowitoliczbowych

`(a)?b:c;` - jeżeli **a** to licz i zwróć **b** wpp licz i zwróć **c**

Przykład

```
int a=3*2;
int b=sizeof a+3,c;
a+=b*3;
int true=1, false=0, w;
w = true && false || 3;
w = a?true:false-nwd(4,2);
w = a?b?a:3,true:false;  //o co tu chodzi?
a=11++ ;                 //?
a=(main(),c=2,3);        // co by było bez ()
c=b=3,a++;
b=++a;
```

fprintf, printf

```
#include <stdio.h>
int main() {
  char znak='e';
  char * napis="Kot \n ma \0 Ale \n.";
  fprintf (stderr,"%d",3);
  fprintf (stdout,"%f",3.0);
  printf ("%c",znak);
  printf ("Ala ma kota");
  printf ("%s",napis);
  fprintf (stderr,napis);
  printf ("%.2f", 3.0124435f);
}            // więcej w man fprintf
```

Instrukcje

- <wyrażenie> **;**
  ;
- **{** <deklaracje> <instrukcje> **}**
- **if (** <wyrażenie> **)** <instrukcja>
  if (<wyrażenie> **)** <instrukcja> **else** <instrukcja>
- **while (** <wyrażenie> **)** <instrukcja>
  do <instrukcja> **while (** <wyrażenie> **)** ;
  for (<wyr1>; <wyr2>; <wyr3>; **)** <instrukcja>
- **continue;**
  break;
  return <wyrażenie>;

switch

`switch (` <wyrażenie> **`)`** {

  `case` <wynik>: <instrukcje>

  `case` <wynik2>: <instrukcje>

...

  `default`: <instrukcje>

}

Uwagi

- W pętli **for** jak nie ma wyr2 to przyjmuje się wartość 1 **`for (;;);  //?`**
- **`continue`** nie dotychyczy **`swich,`** ale **`break`** tak

Ćwiczenie:

- pętla od 0..n
- pętla od 1..n co drugi element
- pętla od n-1 do 0
- pętla od n do 1

Tablice i wskaźniki

```
int a[3];
int *d, b, **c;
int x[10][3];
a[0]=a[1]=[2] =3;
d= &b;
a[0]=3+ *a;
x[3][0]=1;
```

- Tablice n-elementowe są numerowane od 0 do n-1

Ćwiczenie

Napisz program:

- rozwiązujący równanie kwadratowe
- obliczjący nwd
- wypisujący ciąg {2^i} dla i=1..12
- liczący rekurencyjnie symbol Newtona

Przekazywanie struktury "do zmiany"

```
int szescian(int *w) {
  return *w*=*w**w;
}
int a, *b, c;
b= &c;
scanf("%d\n", &a);
scanf("%d\n", b);
szescian(&c);
a=do6potegi(&c);
```

Struktury

```
struct Punkt {
  int x,y,z;
  int kolor;
} a, *b;
struct Punkt g = {1,2,3, 123};

void f (struct Punkt *p){
  (*p).x ++;
  *p.z --;
  p->y = 0;
}
```

Definicje typów

```
struct Punkt g  = { 1, 1 ,1 , 12};
typedef struct Punkt Punkt;
int main() {
  Punkt p1, *p2;
  p1=g;
  p2=&p1;
}
typedef struct Punkt TablicaPunktow[];
typedef struct Punkt *TablicaPunktow2;
typedef struct Punkt **TablicaWskPunktow;
typedef int *(*T)[];
```

Lista

```
typedef struct ListaInt{
  int wartosc;
  struct ListaInt *nastepny;
} ListaInt;
```

Napisz:

- listę dwukierunkową (wypisywanie, dodawanie elementów)
- drzewo (tworzenie z zapisu nawiasowego, dfs)
- AVL

Dynamiczne przydzielanie pamięci

- **man malloc**
  - `l=(*Lista) malloc (sizeof(Lista));`
- **man free**
  - `free(l);`