Programowanie C/C++
Język C - funkcje, tablice i wskaźniki
wykład II

dr Jarosław Mederski

uzupełnienie notatek: dr Jerzy Białkowski

Spis

Funkcje

```
typ nazwa ( lista-parametrów )
{
deklaracje
instrukcje
}
```

Funkcje

```
typ nazwa ( lista-parametrów )
{
deklaracje
instrukcje
}
```

- return `wyrażenie` - przekazuje wartość wyrażenia typu `typ`

Funkcje

```
typ nazwa ( lista-parametrów )
{
deklaracje
instrukcje
}
```

- return `wyrażenie` - przekazuje wartość wyrażenia typu `typ`
- `typ`=void oznacza brak typu

Funkcje

```
typ nazwa ( lista-parametrów )
{
deklaracje
instrukcje
}
```

- return `wyrażenie` - przekazuje wartość wyrażenia typu `typ`
- `typ`=void oznacza brak typu
- `lista-parametrów` - parametry przekazywane są przez wartość

Funkcje

```
typ nazwa ( lista-parametrów )
{
deklaracje
instrukcje
}
```

- return `wyrażenie` - przekazuje wartość wyrażenia typu `typ`
- `typ`=void oznacza brak typu
- `lista-parametrów` - parametry przekazywane są przez wartość
- Funkcje mogą być wywoływane rekurencyjnie

```c
#include <stdio.h>

double max(double a, double b) {
  if (a > b)
    return a;
  else
    return b;
}

int main() {
  double wynik;

  wynik = max(200,300);
  printf("%f\n",wynik);

  return 0;
}
```

```c
#include <stdio.h>
/* deklaracja funkcji */
double max(double, double);

int main() {
  double wynik;

  wynik = max(20.34,30.20);
  printf("%f\n",wynik);

  return 0;
}


double max(double a, double b){
  if (a > b)
    return a;
  else
    return b;
}
```

Przekazywanie parametrów, zasięg zmiennych

```
1 #include <stdio.h>
2
3 void fun(int a){
4   a++;
5   printf("Wartosc a wewnatrz funkcji %d\n",a);
6 }
7
8 int main() {
9   int a=0;
10
11  fun(a);
12  printf("Wartosc a poza funkcja %d\n",a);
13
14  return 0;
15 }
```

Brak argumentów w funkcji

```
1 #include <stdio.h>
2
3 void gwiazdki(void){
4   int i;
5   for(i=0;i<100;i++) {
6     printf("*");
7   }
8   printf("\n");
9 }
10
11 int main() {
12
13   gwiazdki();
14
15   return 0;
16 }
```

Rekurencja

Rekurencja albo rekursja (ang. recursion, z łac. recurrere, przybiec z powrotem) to odwoływanie się funkcji do samej siebie.

$$n! = \begin{cases} 1, & n = 0 \\ n * (n-1)!, & n \geqslant 1 \end{cases}$$

```c
#include <stdio.h>

int silnia(int n){
  if (n==0) {
    return 1;
  else
    return n * silnia(n-1); /* rekurencja */
}

int main() {
  int n, wynik;

  printf("Podaj liczbe ");
  scanf("%d",&n);
  printf("%d! = %d\n",n, silnia(n));

  return 0;
}
```

Algorytm Euklidesa - rekurencja

$NWD(a, b)$

- Jeśli $a = b$, to $NWD(a, b) = a$
- Jeśli $a > b$, to $NWD(a, b) = NWD(a - b, b)$. W przeciwnym przypadku $NWD(a, b) = NWD(a, b - a)$.

```c
#include <stdio.h>

int nwd(int a, int b){
  if ( a == b )
    return a;
  else if ( a > b )
    return nwd(a-b,b);
  else
    return nwd(a,b-a);
}

int main() {
  int a, b;

  scanf("%d %d", &a, &b);
  printf("NWD(%d,%d) = %d\n", a, b, nwd(a,b));

  return 0;
}
```

Deklaracja

```
typ tab[rozm];
```

definiuje tablicę obiektów typu `typ` o rozmiarze `rozm`.
Elementy tej tablicy są indeksowane liczbami od `0` do `rozm-1`.

W przypadku potrzeby używania tablic dwu- lub więcej wymiarowych można deklarować tablice tablic.
Dla przykładu zadeklarowaną w poniższy sposób tablicę

```
typ tab[rozm1][rozm2];
```

można używać jak dwuwymiarową tablicę obiektów typu `typ` o rozmiarze `rozm1`$\times$`rozm2`.

```c
#include <stdio.h>

int main() {
   int i,rozmC;

   int a[5] = {1,2,3};
   int b[5] = {0};
   int c[] = {1,2,3};
   int d[5];
   /* int d[]; otrzymamy blad */

   for(i=0; i<5; i++) {
      d[i]=i*i;
   }

   for(i=0; i<5; i++) {
      printf("%d ",a[i]);
   }
   printf("\n");
```

```
20
21    for(i=0; i<5; i++) {
22      printf("%d ",b[i]);
23    }
24    printf("\n");
25
26    rozmC = sizeof(c)/sizeof(int);
27    for(i=0; i < rozmC; i++) {
28      printf("%d ",c[i]);
29    }
30    printf("\n");
31
32    for(i=0; i<5; i++) {
33      printf("%d ",d[i]);
34    }
35    printf("\n");
36
37    return 0;
38 }
```

```c
#include <stdio.h>
void wyswietl(int tab[][3], int l_w){
/* musimy podac liczbe kolumn - 3 */
   int i,j;
   for(i=0;i<l_w;i++){
      for(j=0;j<3;j++)
         printf("%4d",tab[i][j]);
      printf("\n");
   }
   printf("\n");
}

int main() {
   int a[2][3] = {1,2,3,4,5};
   int b[4][3] = {{1,2},{3,4,5},{1}};
   int c[][3] = {1,2,3,4,5};

   wyswietl(a,2);
   wyswietl(b,4);
   wyswietl(c,2);
   return 0;
}
```

Wskaźnik jest adresem obiektu w pamięci komputera. Deklaracja wskaźnika, który wskazuje na obiekt typu `typ` jest następująca:

```
typ *wsk;
```

Adres obiektu pobieramy za pomocą jednoargumentowego operatora `&`.

```c
#include <stdio.h>

int main() {
   int a = 5, b = 6;
   int *c;

   c = &a; /* c wskazuje na a */
   printf("%d\n", *c);

   b = *c;
   printf("%d\n",b);

   *c = 4;
   printf("%d\n",a);
   printf("%d\n",b);
   printf("%d\n",*c);

   return 0;
}
```

Operatory & i * są wzajemnie odwrotne.

Operatory & i * są wzajemnie odwrotne.

```
1  #include <stdio.h>
2
3  int main() {
4     int a = 5, b = 6;
5     int *c, *d;
6
7     c = &a; /* c wskazuje na a */
8
9     b = *&a;
10    d = &*c;
11
12    printf("%d\n",b);
13    printf("%d\n",*d);
14
15    return 0;
16 }
```

```c
#include <stdio.h>

void fun(int x){
   x++;
   printf("Wewnatrz funkcji fun %d\n",x);
}

void fun_wsk(int *x){
   (*x)++; /* *x++ to nie to samo */
   printf("Wewnatrz funkcji fun_wsk %d\n",*x);
}

int main() {
   int a=5;

   fun(a);
   printf("main %d\n",a);
   fun_wsk(&a);
   printf("main %d\n",a);

   return 0;
}
```

Funkcje w języku C nie są zmiennymi, ale można zdefiniować wskaźniki do funkcji.

```
typ (*funkcja)(...)
```

Uwaga: Powyższa deklaracja różni się od deklaracji `typ *funkcja(...)`, która oznacza „zwykłą“ funkcję zwracającą wskaźnik do obiektu typu `typ`.

```c
#include <stdio.h>


double fun1(double x) {

  return x*x;
}

double wartosc(double (*fp)(double x),
               double a) {
  return  (*fp)(a);
}

int main() {

  printf("%f\n", wartosc(fun1,5));

  return 0;
}
```

```c
#include <stdio.h>
double a;

double *fun1(double x) {
  a=x*x;
  return &a;
}

double wartosc(double *(*fp)(double x),
               double a) {
  return  *(*fp)(a);
}

int main() {

  printf("%f\n", wartosc(fun1,5));

  return 0;
}
```

W języku C istnieje zależność pomiędzy funkcjami a wskaźnikami na funkcje.

W szczególności

- dla funkcji o deklaracji

  ```
  int funkcja();
  ```

  wyrażenie `funkcja()` przyjmuje wartość zwróconą przez rozważaną funkcję, a wyrażenie `funkcja` przyjmuje wartość adresu tej funkcji;

- dla deklaracji

  ```
  double (*fp)(double);
  double a;
  ```

  zamiast wyrażenia `(*fp)(a)` można użyć konstrukcji `fp(a)` .

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

int dodawanie(int a, int b) { return a+b; }
int odejmowanie(int a, int b) { return a-b; }
int (*operacja)(int, int);

int main(int argc, char *argv[]) {
  int liczba1, liczba2;
  char znak;

  if(argc != 4) {
    printf("Zla liczba argumentow!\n");
    printf("Wywolanie: %s liczba1 operacja liczba2\n",
           argv[0]);
    return 1;
  }

  liczba1 = atoi(argv[1]);
  znak = argv[2][0];
  liczba2 = atoi(argv[3]);
```

```
22
23      switch (znak) {
24      case '+':
25        operacja = dodawanie;
26        break;
27      case '-':
28        operacja = odejmowanie;
29        break;
30      default:
31        printf("Niewlasciwy znak operacji!\n");
32        printf("Wywolanie: %s liczba1 operacja ↩
              liczba2\n",
33                argv[0]);
34        return 2;
35    }
36
37    printf("Wynik operacji: %d %c %d = %d\n",
38           liczba1, znak, liczba2,
39           operacja(liczba1, liczba2));
40
41    return 0;
42  }
```

W języku C występuje ścisła zależność pomiędzy tablicami i wskaźnikami. Dla danej tablicy `typ tab[5]`, `tab` jest wskaźnikiem do piewszego (o indeksie 0) elementu tablicy, zaś jego wartość wynosi `*tab`. Ogólnie, `tab[i]` jest równoważne `*(tab+i)`, a stąd `&tab[i]` jest równoważne `(tab+i)`.

```
1  #include <stdio.h>
2
3  int main() {
4     int tab[]={3,4,5,6,7};
5     int *p;
6
7     p=tab;
8     printf("%d\n", *p);
9     p++;
10    printf("%d\n", *p);
11    printf("%d\n", *(tab+4));
12
13    return 0;
14 }
```

```c
#include <stdio.h>
void wyswietl(int tab[][3], int l_w) {
/* musimy podac liczbe kolumn - 3 */
  int i,j;
  for(i=0;i<l_w;i++) {
    for(j=0;j<3;j++)
      printf("%4d",tab[i][j]);
    printf("\n");
  }
  printf("\n");
}
int main() {
  int a[2][3] = {1,2,3,4,5,6};
  wyswietl(a,2);
  printf("wsk %d, tab %d\n", **a, a[0][0]);
  printf("wsk %d, tab %d\n", **(a+1),a[1][0]);
  printf("wsk %d, tab %d\n", **(a+2),a[2][0]);
      /* poza tablica */
  printf("wsk %d, tab %d\n",
          *(*(a+1)+2), a[1][2]);
  return 0;
}
```

Napisy

Stała napisowa w języku C jest tablicą znaków (`char`) zakończoną znakiem `'\0'`. Przykładem jest parametr `"Witaj"` w funkcji

```
printf("Witaj");
```

```c
#include <stdio.h>

void wyswietl(char *);

int main() {
   char *nap = "Witaj";
   char tab[] = "Witoj";
   printf("%c\n\n", nap[2]);
   wyswietl(nap);
   wyswietl(tab);
   tab[3]='i'; /* Blad! nap[3]='i' */
   wyswietl(tab);
   return 0;
}

void wyswietl(char *n){
   while ( *n != '\0' ) {
      printf("%c ", *n);
      n++;
   }
   printf("\n");
}
```

Kopiowanie napisów

Stała napisowa w języku C jest tablicą znaków (`char`) zakończoną znakiem `'\0'`. Przykładem jest parametr `"Witaj"` w funkcji

```
char nap1="Ala ma kota";
char nap2="Witaj";
```

Instrukcja `nap1=nap2;` kopiuje tylko wskaźniki.

```c
#include <stdio.h>

void copy(char [], char []);
void copy2(char *, char *);

int main() {
   char nap1[] = "Ala ma kota";
   char nap2[] = "Witaj";

   printf("%s\n", nap1);

   copy(nap1,nap2);
   printf("%s\n", nap1);

   copy2(nap1,"Hej");
   printf("%s\n", nap1);

   return 0;
}
```

```
23 void copy(char n1[], char n2[]) {
24   int i=0;
25   while( n1[i] != '\0' ) {
26     n1[i]=n2[i];
27     i++;
28   }
29 }
30
31 void copy2(char *n1, char *n2) {
32   while( (*n1++ = *n2++) != '\0' ) {
33     ;
34   }
35 }
```

Argumenty funkcji main()

Funkcja `main()` przyjmuje dwa argumenty: liczbę parametrów oraz tablice parametrów wywołania programu.

```
1  #include <stdio.h>
2
3  int main(int argc, char *argv[]) {
4    printf("Liczba parametrow %d\n", argc);
5    while (argc > 0) {
6      argc--;
7      printf("%s ", argv[argc]);
8    }
9    printf("\n");
10   return 0;
11 }
```

```c
#include <stdio.h>

int main(int argc, char **argv) {
  printf("Liczba parametrow %d\n", argc);
  while (argc--) {

    printf("%s ", *argv++);
  }
  printf("\n");
  return 0;
}
```

```
1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int dodawanie(int a, int b) { return a+b; }
5  int odejmowanie(int a, int b) { return a-b; }
6  int (*operacja)(int, int);
7
8  int main(int argc, char *argv[]) {
9    int liczba1, liczba2;
10   char znak;
11
12   if(argc != 4) {
13     printf("Zla liczba argumentow!\n");
14     printf("Wywolanie: %s liczba1 operacja ↩
         liczba2\n",
15              argv[0]);
16     return 1;
17   }
18
19   liczba1 = atoi(argv[1]);
20   znak = argv[2][0];
21   liczba2 = atoi(argv[3]);
```

```
22
23      switch (znak) {
24      case '+':
25        operacja = dodawanie;
26        break;
27      case '-':
28        operacja = odejmowanie;
29        break;
30      default:
31        printf("Niewlasciwy znak operacji!\n");
32        printf("Wywolanie: %s liczba1 operacja ↩
              liczba2\n",
33                argv[0]);
34        return 2;
35    }
36
37    printf("Wynik operacji: %d %c %d = %d\n",
38           liczba1, znak, liczba2,
39           operacja(liczba1, liczba2));
40
41    return 0;
42  }
```

Problemy

Zadanie

Wczytać liczbę naturalną n i wyznaczyć n liczb ciągu Fibonacciego

$$F_n = \begin{cases} 1, & n = 0, 1 \\ F_{n-1} + F_{n-2}, & n \geqslant 1 \end{cases}$$

Problemy

Zadanie

Wczytać liczbę naturalną n i wyznaczyć n liczb ciągu Fibonacciego

$$F_n = \begin{cases} 1, & n = 0, 1 \\ F_{n-1} + F_{n-2}, & n \geqslant 1 \end{cases}$$

Zadanie

Zdefiniować funkcję

```
double calka (double (*fp)(double x),
              double a, double b, int n),
```

która dla danego wskaźnika do funkcji `fp` *oblicza całkę funkcji* `fp` *na przedziale* $[a, b]$ *metodą trapezów uwzględniając podział odcinka* `n`. *Oblicz całki kilku wybranych funkcji. Porównaj przykład na stronie 26.*

Dziękuję za uwagę.