

# Вказівники на функції

В мові С ім'я функції є константним вказівником на перший байт виконавчого коду функції. Це адреса оперативної пам'яті, яка відповідає *точці входу* даної функції. У разі виклику функції зчитується перша команда за цією адресою, а далі всі наступні команди. Адресу функції можна присвоїти вказівнику та використовувати його для звертання до функції.

Оголошення вказівника на функцію  
*тип\_значення\_функції (\*ім'я\_вказівника)(список  
типів\_параметрів\_функції)*

Операція – “функція” має вищий пріоритет ніж операція \* -  
”вказівник”, тому конструкцію *\*ім'я\_вказівника* необхідно охопити дужками. Інакше дане оголошення було би прототипом функції, яка використовує відповідні параметри і повертає значення, яке є вказівником.

Приклад вказівника на функцію, що має два параметри і повертає вказівник на дані з типом *char*

*char\* (\*pfun) (char\*, unsigned)*

Якщо оголошено дві функції

*char\* FindWord (char\* st, unsigned num);*

*char\* DelWord (char\* sent, unsigned k);*

то коректними будуть присвоєння

*pfun = FindWord*      або      *pfun = &FindWord*

*pfun = DelWord*      або      *pfun = &DelWord*

Вказівник можна застосовувати для звертання до функції.

Після першого присвоєння, наступне звертання *(\*pfun)(str, 3);*

рівнозначне виклику *FindWord (str, 3);*

Можна використовувати спрощене звертання *pfun (str, 3),*

але краще конструкцію з розадресованим вказівником.

## Приклад

```
#include<stdio.h>
#include<conio.h>
void print(char *s)
{
    puts(s);
}
void main(void)
{
    void (*efct)(char *s);
    efct=&print; /* efct=print */
    (*efct)("Function Print!"); /* efct("Function Print!"); */
}
```

Показчикам на функції можна присвоювати адреси стандартних бібліотечних функцій.

```
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
void main(void)  
{  
    double (*fn)(double);  
    float y,x=1;  
    fn=sin;  
    y=fn(x);  
    printf("sin(%g)==%g\n",x,y);  
    fn=cos;  
    y=fn(x);  
    printf("cos(%g)==%g\n",x,y);  
}
```

Найчастіше вказівники на функції використовуються як формальні параметри у функціях вищого рівня. Це дає змогу створювати функції, які використовують інші функції без огляду на їх конкретні імена та внутрішнє наповнення.

Приклад

```
#include<stdio.h>  
#include<conio.h>  
#include<math.h>  
double fn(double (*pfn)(double ), double x)  
{  
    double y=pfn(x);  
    printf("y==%g\n", y);  
    return y;  
}
```

```
double fun1 (double x)
{
    return sin(x)*cos(x);
}
double fun2 (double x)
{
    if (x>=0)
        return 3*cos(1.5*x)
    else
        return cos(x)*cos(x);
}
void main(void)
{
    fn(sin,1);
    fn(fun1,1);
    fn(&fun2,1);
}
```

*Приклад.* Знайти додатний корень трансцендентного рівняння  $\exp(x) - 2 - x = 0$  з похибкою  $EPS$

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

```
#include <math.h>
```

```
#define EPS 1e-10 // точність результату
```

```
double f (double x);
```

```
int main()
```

```
{ double l = 0, r = 2, c;
```

```
while( r - l > EPS )
```

```
{ c = ( l + r ) / 2; //середина проміжку
```

```
if( f(c) * f(r) < 0 ) //визначаємо, в якій з частин
```

```
l = c; // знаходиться корень
```

```
else r = c; }
```

```
printf ("% .10lf\n", (l + r)/2 );
```

```
}
```

```
double f (double x)  
  {  
    return exp(x) - 2 - x;  
  }
```

Запрограмовано алгоритм ділення навпіл. Корень знаходиться між  $l = 0$  та  $r = 2$ . Знаходимо середину  $c$  відрізка  $[l, r)$ . Корень буде на одному з відрізків : на  $[l, c)$ , або на  $[c, r)$ , там, де значення функції на кінцях мають різні знаки (теорема Ролля). Обираємо потрібний з двох відрізків та повторюємо алгоритм. Виконуємо ділення навпіл, поки довжина відрізка не стане менше заданої точності.



В прикладі обчислювався нуль конкретної функції  $f(x) = \exp(x) - 2 - x$ . Запишемо функцію, яка знаходить нулі довільних функцій. Для цього потрібно передавати функцію в якості аргумента.

```
#include <stdio.h>  
#include <math.h>  
#define EPS 1e-16  
double root (double l, double r, double (*f)(double));  
double f1 (double x);  
double f2 (double x);
```

```

int main()
{   printf ("root1 = %lf\n", root(0, 2, f1)); // обчислюємо
                                             // корень  $f1(x) = 0$ 
printf ("root2 = %lf\n", root(0, 2, f2)); // обчислюємо
                                             // корень  $f2(x) = 0$ 

return 0;   }
double root(double l, double r, double (*f)(double))
{   double c;
      while( r - l > EPS )   {   c = ( l + r ) / 2;
          if( f(c) * f(r) < 0 )   l = c;
          else   r = c;   }

return l;
}
double f1 (double x) {   return cos(x) - 3 * x;   }
double f2 (double x) {   return exp(x) - x - 2;   }

```