

Function Pointer

A function pointer in C is a pointer that stores the address of a function. It allows you to call a function indirectly through its address, instead of using the function name directly.

→ Function pointers are commonly used to invoke callback functions.

Syntax:- return type . (*function ptr name) (Arg 1, Arg 2, ...)

Example:- `int (*ptr) (int, int);`

1. `int` → return type (function returns int)
2. `(*ptr)` → ptr is a function pointer
3. `(int, int)` → function accepts two integers

Read → ptr is a function pointer to a function which returns integer value & accepts two integer arguments.

Program 1:-

```
int add (int a, int b) ← 1000
{
    return a+b;
}

int main()
{
    int x=5;
    int y=10;
    int sum;
    // sum = add(x,y);

    int (*ptr) (int, int); // Declaration
    ptr = add;             // or ptr = &add

    sum = ptr(x,y);
    printf(" %d", sum);
    return 0;
}
```

8B
ptr [1000]

①

```
void sum (int, int) ✓
{
    int main()
    {
        sum(5,10); ✓
    }

    void sum (int a, int b) ✓
    {
        printf("sum %d", a+b);
    }
}
```

OR

②

```
void sum (int a, int b)
{
    printf("sum %d", a+b);
}

int main()
{
    sum(5,10); // calling by name
    return 0;
}
```

Note:- if we have function definition available before function call then declaration / prototype is optional.

Program 2:- subtraction using function pointer

```
int subtract (int a, int b)
{
    return a-b;
}

int main()
{
    int x=5;
    int y=10;
    int sub;

    int (*ptr) (int, int);
    ptr = subtract;
```

```

sub = ptr(x, y);      5-10 = -5
printf("%d", sub);    → -5
return 0;
}

```

==x==

Array of function pointers :- We can create array which can store address of n number of function.

Syntax:-

```
int (*ptr[]) (int, int);
```

```

int add(int a, int b) // address 1000 ←
{
    return (a+b);
}

```

```

int sub(int a, int b) // address 2000 ←
{
    return (a-b);
}

```

```

int mult(int a, int b) // address 3000 ←
{
    return (a*b);
}

```

= int (*ptr[]) (int, int) = { add, sub, mult };

ptr [1000 | 2000 | 3000]
 0 1 2

printf("%d", ptr[0](5, 10)); → add(5, 10) ⇒ 15

printf("%d", ptr[1](5, 10)); → sub(5, 10) ⇒ -5

printf("%d", ptr[2](5, 10)); → mult(5, 10) ⇒ 50