

Pointers in Functions

☰ Tags	c functions pointers
🕒 Created time	@October 8, 2024 10:57 AM
# Lecture No.	8
☑ Reviewed	<input type="checkbox"/>

Pass by reference

❓ Swapping two elements in an array is an important step of many sorting algorithm. How can we write a function to do so?

```
#include<stdio.h>

void swap_with_pointer(int *a, int *b);

int main(int argc, char *argv[])
{
    int num1 = 5;
    int num2 = 6;

    int *pNum1 = NULL; /* initialise a pointer with NULL, i.e, a
    int *pNum2 = 0; /* Another way of initialise a pointer to 0

    printf("Before swap : num1 = %d, num2 = %d \n", num1, num2);

    pNum1 = &num1;
    pNum2 = &num2;
```

```

    swap_with_pointer(pNum1,pNum2);
    printf("After swap : num1 = %d, num2 = %d \n", num1, num2);

    return 0;
}

/*
Here we are using pass-by variables, i.e a copy of the variable is passed.
Changing a and b here will only change them locally in the function.
void wrong_swap(int a, int b) {
    int temp;
    temp = a;
    a = b;
    b = temp;
}
*/

void swap_with_pointer(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

-----
$ ./swap.out
Before swap : num1 = 5, num2 = 6
After swap : num1 = 6, num2 = 5

```

Bubble Sort Example

```

#include<stdio.h>

void swap(int *a, int *b);

int main(int argc, char *argv[])
{
    int a[5] = {5, 3, 2, 4, 1};

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

    printf("\n ^ Before Sort \n");

    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            if (a[i] > a[j]) {
                // use & here as the reference operator to get the address of the element
                swap(&a[i], &a[j]);
            }
        }
    }

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

    printf("\n ^ After Sort \n");

    return 0;
}

```

```

}

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

```

```

-----
$ ./bubblesort.out
5, 3, 2, 4, 1,
  ^ Before Sort
5, 4, 3, 2, 1,
  ^ After Sort

```

Return more than one value with different types?

? Write a function to find the max number in an array, returning this and its position.

```

#include <stdio.h>

void findMax(float a[], int length, float *maxVal, int *pos);

int main(int argc, char *argv[])
{
    float maxVal = -999; // setting this to a value outside the
    int pos = 0;

```

```

float a[10] = {0, 1, 2, 4, 2432, 5, 3, 2, 4, 1};

// sizeof(a) - gives the total amount of bytes in the array
// we need to divide by the size of one element (sizeof(a[0])
// => sizeof(a)/sizeof(a[0]) - gives the length of the array

findMax(a, sizeof(a)/sizeof(a[0]), &maxVal, &pos);
printf("Max value is %.1f at position %d \n", maxVal, pos);

return 0;
}

void findMax(float a[], int length, float *maxVal, int *pos) {
    *maxVal = a[0];
    for (int i = 1; i < length; i++)
    {
        if (a[i] > *maxVal) {
            *maxVal = a[i];
            *pos = i;
        }
    }
}

-----
$ ./maxAndPos.out
Max value is 2432.0 at position 4

```

Pointer to a Function

Similarly with variables, a pointer can be used to point to a function. You can declare a function pointer as follows:

```
type (*pFunctionName) (types_of_params);
```

In C, similar to array names, function names refer to the address of the function. So we can assign the value of a function pointer as follows:

```
pFunctionName = functionName
```

The * is not needed here.

Example

```
int sum(int, int);
int product(int, int);
int difference(int, int);

int main(void)
{
    int a = 10;                // Initial value for a
    int b = 20;                // Initial value for b
    int result = 0;            // Storage for results
    int (*pfunction)(int, int); // Function pointer

    pfunction = sum;           // Points to function sum
    result = pfunction(a, b);   // Call sum() through pointer
    printf("pfunction = sum result = %2d\n", result);

    pfunction = product;       // Points to function product
    result = pfunction(a, b);   // Call product() through pointer
    printf("pfunction = product result = %2d\n", result);

    pfunction = difference;     // Points to function difference
    result = pfunction(a, b);   // Call difference() through pointer
    printf("pfunction = difference result = %2d\n", result);
}
```

```

        return 0;
    }

    int sum(int x, int y)
    {
        return x + y;
    }
    int product(int x, int y)
    {
        return x * y;
    }
    int difference(int x, int y)
    {
        return x - y;
    }

    -----
$ ./pointerToFunction.out
pfunction = sum result = 30
pfunction = product result = 200
pfunction = difference result = -10

```



Parameter types need to be matched between function pointers and functions.

Pointers to Functions as Arguments

You can also use pointers to point to a function in c. It allows for different functions to be called depending on which function address the pointer is pointing towards.

```

#include <stdio.h>
// Function prototypes
int sum(int,int);
int product(int,int);
int difference(int,int);
int any_function(int(*pfun)(int, int), int x, int y);

int main(void)
{
    int a = 10;                // Initial value for a
    int b = 5;                // Initial value for b
    int result = 0;           // Storage for results
    int (*pf)(int, int) = NULL; // Pointer to function

    // Passing a pointer to a function
    pf = sum;
    result = any_function(pf, a, b);
    printf("result = %2d\n", result );

    pf = product;
    result = any_function(pf,a, b);
    printf("result = %2d\n", result );

    // Passing the address of a function
    printf("result = %2d\n", any_function(difference, a, b));
    return 0;
}

// Definition of a function to call a function
int any_function(int(*pfun)(int, int), int x, int y)
{
    return pfun(x, y);
}

// Definition of the function sum

```



```

int sum(int x, int y)
{
    return x + y;
}

// Definition of the function product
int product(int x, int y)
{
    return x * y;
}

// Definition of the function difference
int difference(int x, int y)
{
    return x - y;
}

```

The any function has 3 parameters:

1. A pointer to a function that accepts two integer arguments and returns an integer.
- 2&3. Will be used in the call to the function given by the first parameters.