

Tutorial 9

Pointers in C

Shuyue Hu

Content

- **Basic concept of pointers**
- **Pointer arithmetic**
- **Array of pointers**
- **Pointer to pointer**
- **Passing pointers to functions in C**
- **Return pointer from functions in C**

Q4: What is the difference between passed by value and passed by reference?

- **Passed By Value:** The actual arguments are **copied** to the formal arguments, hence any operation performed by function on arguments **doesn't affect** actual parameters.
- **Passed By Reference:** The **addresses** of actual arguments are passed to the formal parameters, which means any operation performed on formal parameters **affects** the value of actual parameters.
- C parameters are always **passed by value** rather than by reference. However, if you think of the **address** of an object as being a **reference** to that object, then you can pass that reference by value. Because you're passing **the value** of the pointer to the method and then dereferencing it to get the integer that is pointed to.
- Reference: <http://stackoverflow.com/questions/2229498/passing-by-reference-in-c>

Example

```
#include <stdio.h>
void Func1(int, int); // pass by value
void Func2(int *, int *); // pass by reference

int main( )
{
    int a = 8, b = 9;
    printf("Before Func1 is called, a = %d, b = %d\n", a, b);
    Func1(a, b);
    printf("After Func1 is called, a = %d, b = %d\n\n", a, b);

    printf("\nBefore Func2 is called, a = %d, b = %d\n", a, b);
    Func2(&a, &b);
    printf("After Func2 is called, a = %d, b = %d\n\n", a, b);
}
```

Example

```
void Func1(int a, int b)
{
    a = 0;
    b = 0;
    printf("The value inside Func1, a = %d, b = %d\n", a,
b);
}
```

```
void Func2(int *pa, int *pb)
{
    *pa = 0;
    *pb = 0;
    printf("The value inside Func2, *pa = %d, *pb =
%d\n", *pa, *pb);
}
```

Content

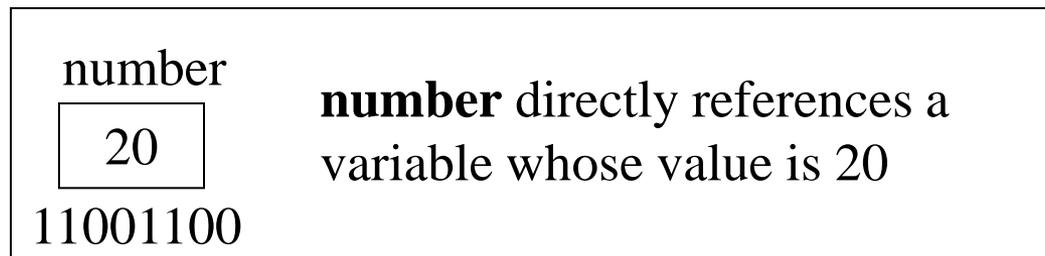
- **Basic concept of pointers**
- Pointer arithmetic
- Array of pointers
- Pointer to pointer
- Passing pointers to functions in C
- Return pointer from functions in C

What is a pointer

- So far, we have seen that a variable is used to store a **value**.
- Variables allow the programmer to directly manipulate the data in memory.
- A pointer variable, however, does not store a value but store the **address of the memory** space which contain the value i.e. **it directly points to a specific memory address.**
- Why would we want to use pointers?
 - To call a function by reference so that the data passed to the function can be changed inside the function.
 - To create a dynamic data structure which can grow larger or smaller as necessary.

Variable declaration

- A variable declaration such as,
 - ***int number = 20;*** causes the compiler to allocate a memory location for the variable *number* and store in it the integer value 20.
 - This absolute address of the memory location is readily available to our program during the run time.
 - The computer uses this address to access its content.



Pointer declaration

- General Format:

*data_type *pointer_name;*

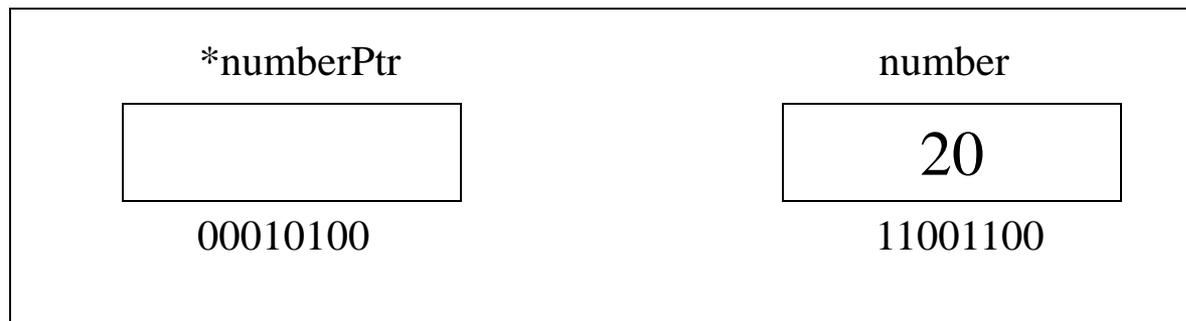
- A pointer declaration such as,

*int *numberPtr;*

- declares *numberptr* as a variable that **points to an integer variable**. Its content is a **memory address**.
- The asterisk ***** indicates that the variable being declared is a **pointer** variable instead of a normal variable.

Pointer declaration (cont.)

- Consider the following declaration
*int *numberPtr, number = 20;*
- In this case, two memory address have been reserved in the memory, namely the *numberPtr* and *number*.
- The value in variable *number* is of type integer, and the value in variable *numberPtr* is an address for another memory.



Pointer Initialization

- To prevent the pointer from pointing to a random memory address, it is advisable that the pointer is initialized to **0**, **NULL** or **an address before being used**.
- A pointer with the value **NULL**, points to nothing.
- Initializing a pointer to **0** is equivalent to initializing a pointer to **NULL**, but **NULL** is preferred.

Pointer Operator (& and *)

- When a pointer is created, it is not pointing to any valid memory address. Therefore, we need to assign it to a variable's address by using the ampersand **&** operator. This operator is called a **reference operator**.
- Look at this example:

```
int number = 20;  
int *numberPtr; // (a) We define a pointer variable  
numberPtr = &number; // (b) assign the address of a variable to a pointer  
printf("number = %d", *numberPtr); // (c) finally access the value at the  
address available in the pointer variable.
```

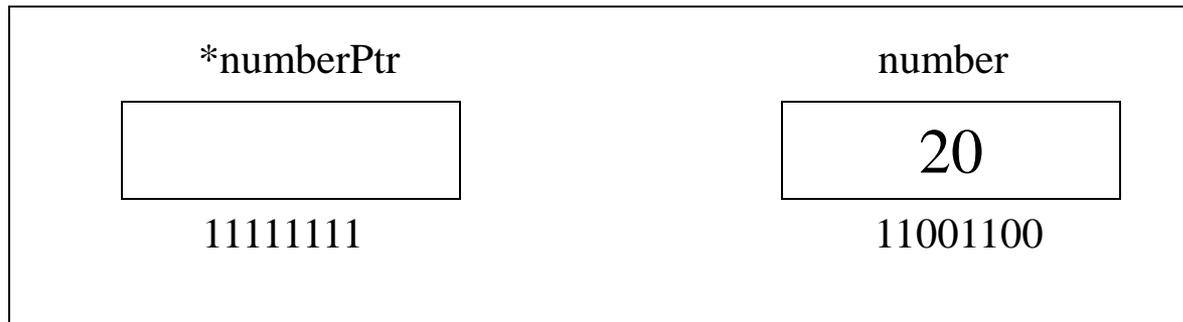
Output:

number = 20

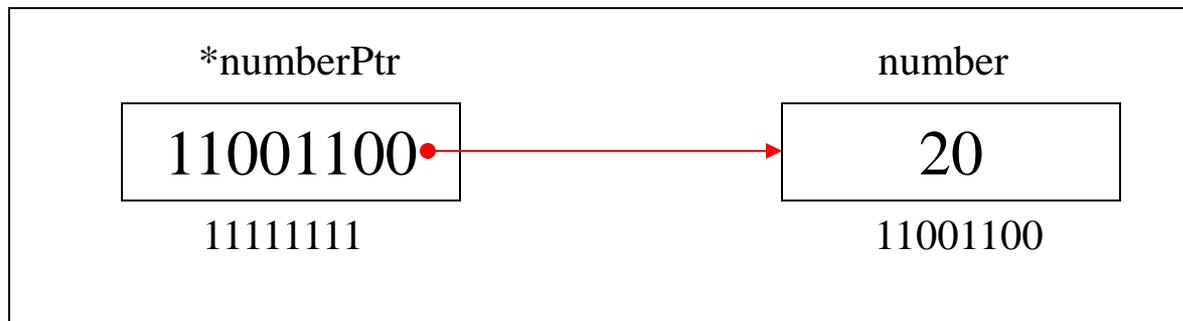
- The statement **numberPtr = &number** assigns the address of the variable number to a pointer variable numberPtr.
- Variable numberPtr is then said as to “**point to**” variable number.

Graphical representation

- `int *numberPtr, number = 20;`



- `numberPtr = &number;`



Pointer Operator (& and *) (cont.)

- After a pointer is assigned to a particular address, the value in the pointed address can be accessed/modified using the asterisk ***** operator.
- This operator is commonly called as the ***indirection operator*** or ***dereferencing operator***.
- The ***** operator returns the value of the object to which its operand points. For example, the statement
 - `printf("number = %d", *numberPtr);`
`//prints the value of variable number, namely as 20.`
`//Using * in this manner is called dereferencing operator.`

Example: & and *

```
#include <stdio.h>
int main( )
{
    int var = 10;
    int *ptrvar = &var;

    printf("The address of the variable var is: %d\n", &var);
    printf("The value of the pointer ptrvar is: %d\n", ptrvar);
    printf("Both values are the same\n\n");

    printf("The value of the variable var is: %d\n", var);
    printf("The value of *ptrvar is: %d\n", *ptrvar);
    printf("Both values are the same\n\n");

    printf("The address of the value pointed by ptrvar is: %d\n", &*ptrvar);
    printf("The value inside the address of ptrvar is: %d\n", *&ptrvar);
    printf("Both values are the same\n\n");
}
```

Example: & and *

```
/*Sample Output */
```

```
The address of the variable var is: 1245052  
The value of the pointer ptrvar is: 1245052  
Both values are the same
```

```
The value of the variable var is: 10  
The value of *ptrvar is: 10  
Both values are the same
```

```
The address of the value pointed by ptrvar is: 1245052  
The value inside the address of ptrvar is: 1245052  
Both values are the same
```

```
Press any key to continue
```

&* and *&

- & and * are inverse operations.
- &* acts equivalent to *& and this leads back to the original value.
- Example: (Assume that the address of num is 1245052)

```
#include <stdio.h>
int main()
{
    int num = 5;
    int *numPtr = &num;
    printf("%d \n", numPtr);
    printf("%d \n", &*numPtr);
    printf("%d \n", *&numPtr);
}
```

Output:
1245052
1245052
1245052

Content

- Basic concept of pointers
- **Pointer arithmetic**
- Array of pointers
- Pointer to pointer
- Passing pointers to functions in C
- Return pointer from functions in C

Pointer arithmetic

- A pointer in c is an address, which is a numeric value. Therefore, you can perform arithmetic operations on a pointer.
- There are four arithmetic operators that can be used on pointers: ++, --, +, and –
- `int *ptr = 1000;` `char *ptr = 1000;`
- `ptr++` `ptr++`
- `ptr = 1004` `ptr = 1001`

Example

```
#include <stdio.h>
const int MAX = 3;
int main () {
    int var[] = {10, 100, 200};
    int i, *ptr;
    /* let us have array address in pointer */
    ptr = var;
    for ( i = 0; i < MAX; i++) {
        printf("Address of var[%d] = %x\n", i, ptr );
        printf("Value of var[%d] = %d\n", i, *ptr );
        /* move to the next location */
        ptr++;
    }
    return 0;
}
```

Content

- Basic concept of pointers
- Pointer arithmetic
- **Array of pointers**
- Pointer to pointer
- Passing pointers to functions in C
- Return pointer from functions in C

Array of pointers

- declaration of an array of pointers to an integer:

```
int *ptr[MAX];
```

- It declares *ptr* as an array of MAX integer pointers. Thus, each element in *ptr*, holds a pointer to an **int** value.

Example

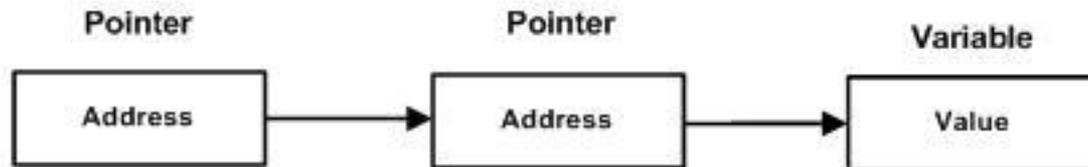
```
#include <stdio.h>
const int MAX = 3;
int main () {
    int var[] = {10, 100, 200};
    int i, *ptr[MAX];
    for ( i = 0; i < MAX; i++) {
        ptr[i] = &var[i]; /* assign the address of
integer. */
    }
    for ( i = 0; i < MAX; i++) {
        printf("Value of var[%d] = %d\n", i, *ptr[i]
);
    }
    return 0;
}
```

Content

- Basic concept of pointers
- Pointer arithmetic
- Array of pointers
- **Pointer to pointer**
- Passing pointers to functions in C
- Return pointer from functions in C

Pointer to Pointer

- When we define *a pointer to a pointer*, the **first** pointer contains the **address** of the second pointer, which points to the location that contains the actual value as shown below.



- Declare a pointer to a pointer of type int

int **var;

Example

```
#include <stdio.h>
int main () {
    int var;
    int *ptr;
    int **pptr;

    var = 3000;
    /* take the address of var */
    ptr = &var;
    /* take the address of ptr using address of operator & */
    pptr = &ptr;

    /* take the value using pptr */
    printf("Value of var = %d\n", var );
    printf("Value available at *ptr = %d\n", *ptr );
    printf("Value available at **pptr = %d\n", **pptr);
    return 0;
}
```

Content

- Basic concept of pointers
- Pointer arithmetic
- Array of pointers
- Pointer to pointer
- **Passing pointers to functions in C**
- Return pointer from functions in C

Passing pointers to functions in C

- C programming allows passing a pointer to a function.
- To do so, simply declare the function parameter as a pointer type.
 - Declare the variable that is meant to return a value to the calling function as a pointer variable in the formal parameter list of the function.
`void function_name(int *varPtr);`
 - When to call the function, use a variable together with address operator (&)
`function_name(&var);`

Parameter Passing by Reference/Pointer

- This way of passing the argument can realize the purpose of **passing by reference**. However, there is no “passed by reference” in C.
- Just because you're passing **the value** of the pointer to the method and then dereferencing it to get the integer that is pointed to.
- **When the value referenced by the pointer is changed inside the function, the value in the actual variable will also change.**
- When a pointer is passed to a function, we are actually passing the **address** of a variable to the function.
- Since we have the address, we can **directly manipulate** the data in the address.

Example

```
#include <stdio.h>
void Func1(int, int); // pass by value
void Func2(int *, int *); // pass by pointer

int main( )
{
    int a = 8, b = 9;
    printf("Before Func1 is called, a = %d, b = %d\n", a, b);
    Func1(a, b);
    printf("After Func1 is called, a = %d, b = %d\n\n", a, b);

    printf("\nBefore Func2 is called, a = %d, b = %d\n", a, b);
    Func2(&a, &b);
    printf("After Func2 is called, a = %d, b = %d\n\n", a, b);
}
```

Example

```
void Func1(int a, int b)
{
    a = 0;
    b = 0;
    printf("The value inside Func1, a = %d, b = %d\n", a,
b);
}
```

```
void Func2(int *pa, int *pb)
{
    *pa = 0;
    *pb = 0;
    printf("The value inside Func2, *pa = %d, *pb =
%d\n", *pa, *pb);
}
```

Result

```
/* output */
```

```
Before Func1 is called, a = 8, b = 9
```

```
The value inside Func1, a = 0, b = 0
```

```
After Func1 is called, a = 8, b = 9
```

```
Before Func2 is called, a = 8, b = 9
```

```
The value inside Func2, *pa = 0, *pb = 0
```

```
After Func2 is called, a = 0, b = 0
```

```
Press any key to continue
```

Content

- Basic concept of pointers
- Pointer arithmetic
- Array of pointers
- Pointer to pointer
- Passing pointers to functions in C
- **Return pointer from functions in C**

Return pointer from functions in C

- Declare a function returning a pointer:

```
int * myFunction() { ... }
```

- It is not a good idea to return the address of a local variable outside the function, so you would have to define the local variable as **static** variable.

Example

```
#include <stdio.h>
#include <time.h>
/* function to generate and retrun random numbers. */
int * getRandom( ) {
    static int r[10];
    int i;
    /* set the seed */
    srand( (unsigned)time( NULL ) );
    for ( i = 0; i < 10; ++i) {
        r[i] = rand();
        printf("%d\n", r[i] );
    }
    return r;
}

/* main function to call above defined function */
int main () {
    /* a pointer to an int */
    int *p;
    int i;
    p = getRandom();
    for ( i = 0; i < 10; i++ ) {
        printf("* (p+[%d]) :%d\n",i,* (p + i)
    );
    }
    return 0;
}
```

Summary

- **Some Interview Questions**
- **Basic concept of pointers**
- **Pointer arithmetic**
 - `ptr++`
- **Array of pointers**
 - `int *ptr[MAX];`
- **Pointer to pointer**
 - `int **var;`
- **Passing pointers to functions in C**
 - `void function_name(int *varPtr);`
- **Return pointer from functions in C**
 - `int * myFunction() { ... }`

Thank you!