Data Structures Using C

Second Edition

Reema Thareja

Assistant Professor
Department of Computer Science
Shyama Prasad Mukherjee College for Women
University of Delhi

OXFORD
UNIVERSITY PRESS

Output

```
In main(), a = 1 and b = 2
In function (Call By Value Method) – a = 2 and b = 1
In main(), a = 1 and b = 2
In main(), c = 3 and d = 4
In function (Call By Reference Method) – c = 4 and d = 3
In main(), c = 4 and d = 3
```

1.11 POINTERS

Every variable in C has a name and a value associated with it. When a variable is declared, a specific block of memory within the computer is allocated to hold the value of that variable. The size of the allocated block depends on the data type.
Consider the following statement.

```
int x = 10;
```

When this statement executes, the compiler sets aside 2 bytes of memory to hold the value 10. It also sets up a symbol table in which it adds the symbol `x` and the relative address in the memory where those 2 bytes were set aside.

(Note the size of integer may vary from one system to another. On 32 bit systems, integer variable is allocated 4 bytes while on 16 bit systems it is allocated 2 bytes.)

Thus, every variable in C has a value and also a memory location (commonly known as *address*) associated with it. We will use terms `rvalue` and `lvalue` for the value and the address of the variable, respectively.

The `rvalue` appears on the right side of the assignment statement (10 in the above statement) and cannot be used on the left side of the assignment statement. Therefore, writing `10 = k;` is illegal. If we write,

```
int x, y;
x = 10;
y = x;
```

then, we have two integer variables `x` and `y`. The compiler reserves memory for the integer variable `x` and stores the `rvalue` 10 in it. When we say `y = x`, then `x` is interpreted as its `rvalue` since it is on the right hand side of the assignment operator `=`. Therefore, here `x` refers to the value stored at the memory location set aside for `x`, in this case 10. After this statement is executed, the `rvalue` of `y` is also 10.

You must be wondering why we are discussing addresses and `lvalues`. Actually pointers are nothing but memory addresses. A pointer is a variable that contains the memory location of another variable. Therefore, a pointer is a variable that represents the location of a data item, such as a variable or an array element. Pointers are frequently used in C, as they have a number of useful applications. These applications include:

- Pointers are used to pass information back and forth between functions.
- Pointers enable the programmers to return multiple data items from a function via function arguments.
- Pointers provide an alternate way to access the individual elements of an array.
- Pointers are used to pass arrays and strings as function arguments. We will discuss this in subsequent chapters.
- Pointers are used to create complex data structures, such as trees, linked lists, linked stacks, linked queues, and graphs.

- Pointers are used for the dynamic memory allocation of a variable (refer Appendix A on memory allocation in C programs).

1.11.1 Declaring Pointer Variables

The general syntax of declaring pointer variables can be given as below.

```
data_type *ptr_name;
```

Here, `data_type` is the data type of the value that the pointer will point to. For example,

```
int *pnum;
char *pch;
float *pfnum;
```

In each of the above statements, a pointer variable is declared to point to a variable of the specified data type. Although all these pointers (`pnum`, `pch`, and `pfnum`) point to different data types, they will occupy the same amount of space in the memory. But how much space they will occupy will depend on the platform where the code is going to run. Now let us declare an integer pointer variable and start using it in our program code.

```
int x= 10;
int *ptr;
ptr = &x;
```

In the above statement, `ptr` is the name of the pointer variable. The `*` informs the compiler that `ptr` is a pointer variable and the `int` specifies that it will store the address of an integer variable. An integer pointer variable, therefore, 'points to' an integer variable. In the last statement, `ptr` is assigned the address of `x`. The `&` operator retrieves the `lvalue` (address) of `x`, and copies that to the contents of the pointer `ptr`. Consider the memory cells given in Fig. 1.11.

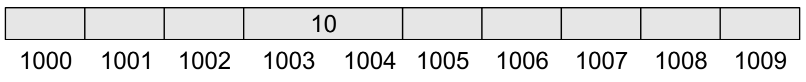

Figure 1.11 Memory representation

Now, since `x` is an integer variable, it will be allocated 2 bytes. Assuming that the compiler assigns it memory locations 1003 and 1004, the address of `x` (written as `&x`) is equal to 1003, that is the starting address of `x` in the memory. When we write, `ptr = &x`, then `ptr = 1003`.

We can 'dereference' a pointer, *i.e.*, we can refer to the value of the variable to which it points by using the unary `*` operator as in `*ptr`. That is, `*ptr = 10`, since 10 is the value of `x`. Look at the following code which shows the use of a pointer variable:

```
#include <stdio.h>
int main()
{
      int num, *pnum;
      pnum = &num;
      printf("\n Enter the number : ");
      scanf("%d", &num);
      printf("\n The number that was entered is : %d", *pnum);
      return 0;
}
```

Output

```
Enter the number : 10
The number that was entered is : 10
```

What will be the value of `*(&num)`? It is equivalent to simply writing `num`.

1.11.2 Pointer Expressions and Pointer Arithmetic

Like other variables, pointer variables can also be used in expressions. For example, if `ptr1` and `ptr2` are pointers, then the following statements are valid:

```
int num1 = 2, num2 = 3, sum = 0, mul = 0, div = 1;
int *ptr1, *ptr2;
ptr1 = &num1;
ptr2 = &num2;
sum = *ptr1 + *ptr2;
mul = sum * (*ptr1);
*ptr2 += 1;
div = 9 + (*ptr1)/(*ptr2) - 30;
```

In C, the programmer may add integers to or subtract integers from pointers as well as subtract one pointer from the other. We can also use shorthand operators with the pointer variables as we use them with other variables.

C also allows comparing pointers by using relational operators in the expressions. For example, `p1 > p2`, `p1 == p2` and `p1 != p2` are all valid in C.

Postfix unary increment `(++)` and decrement `(--)` operators have greater precedence than the dereference operator `(*)`. Therefore, the expression `*ptr++` is equivalent to `*(ptr++)`, as `++` has greater operator precedence than `*`. Thus, the expression will increase the value of `ptr` so that it now points to the next memory location. This means that the statement `*ptr++` does not do the intended task. Therefore, to increment the value of the variable whose address is stored in `ptr`, you should write `(*ptr)++`.

1.11.3 Null Pointers

So far, we have studied that a pointer variable is a pointer to a variable of some data type. However, in some cases, we may prefer to have a *null pointer* which is a special pointer value and does not point to any value. This means that a `null` pointer does not point to any valid memory address.

To declare a null pointer, you may use the predefined constant `NULL` which is defined in several standard header files including `<stdio.h>`, `<stdlib.h>`, and `<string.h>`. After including any of these files in your program, you can write

```
int *ptr = NULL;
```

You can always check whether a given pointer variable stores the address of some variable or contains `NULL` by writing,

```
if (ptr == NULL)
{
      Statement block;
}
```

You may also initialize a pointer as a null pointer by using the constant 0

```
int *ptr,
ptr = 0;
```

This is a valid statement in C as `NULL` is a preprocessor macro, which typically has the value or replacement text 0. However, to avoid ambiguity, it is always better to use `NULL` to declare a null pointer. A function that returns pointer values can return a null pointer when it is unable to perform its task.

1.11.4 Generic Pointers

A generic pointer is a pointer variable that has *void* as its data type. The *void pointer*, or the generic pointer, is a special type of pointer that can point to variables of any data type. It is declared like a normal pointer variable but using the `void` keyword as the pointer's data type. For example,

```
void *ptr;
```

In C, since you cannot have a variable of type `void`, the void pointer will therefore not point to any data and, thus, cannot be dereferenced. You need to cast a void pointer to another kind of pointer before using it.

Generic pointers are often used when you want a pointer to point to data of different types at different times. For example, take a look at the following code.

```
#include <stdio.h>
int main()
{
      int x=10;
      char ch = ‘A’;
      void *gp;
      gp = &x;
      printf("\n Generic pointer points to the integer value = %d", *(int*)gp);
      gp = &ch;
      printf("\n Generic pointer now points to the character= %c", *(char*)gp);
      return 0;
}
```

Output

```
Generic pointer points to the integer value = 10
Generic pointer now points to the character = A
```

It is always recommended to avoid using void pointers unless absolutely necessary, as they effectively allow you to avoid type checking.

Programming Example

9. Write a program to add two integers using pointers and functions.

```
#include <stdio.h>
void sum (int*, int*, int*);
int main()
{
        int num1, num2, total;
        printf("\n Enter the first number : ");
        scanf("%d", &num1);
        printf("\n Enter the second number : ");
        scanf("%d", &num2);
        sum(&num1, &num2, &total);
        printf("\n Total = %d", total);
        return 0;
}
void sum (int *a, int *b, int *t)
{
        *t = *a + *b;
}
```

Output

```
Enter the first number : 23
Enter the second number : 34
Total = 57
```

1.11.5 Pointer to Pointers

In C, you can also use pointers that point to pointers. The pointers in turn point to data or even to other pointers. To declare pointers to pointers, just add an asterisk `*` for each level of reference.

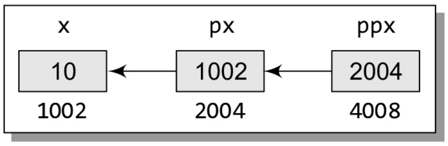

Figure 1.12 Pointer to pointer

For example, consider the following code:

```
int x=10;
int *px, **ppx;
px = &x;
ppx = &px;
```

Let us assume, the memory locations of these variables are as shown in Fig. 1.12.

Now if we write,

```
printf("\n %d", **ppx);
```

Then, it would print 10, the value of `x`.

1.11.6 Drawbacks of Pointers

Although pointers are very useful in C, they are not free from limitations. If used incorrectly, pointers can lead to bugs that are difficult to unearth. For example, if you use a pointer to read a memory location but that pointer is pointing to an incorrect location, then you may end up reading a wrong value. An erroneous input always leads to an erroneous output. Thus however efficient your program code may be, the output will always be disastrous. Same is the case when writing a value to a particular memory location.

Let us try to find some common errors when using pointers.

```
int x, *px;
x=10;
*px = 20;
```

Error: Un-initialized pointer. `px` is pointing to an unknown memory location. Hence it will overwrite that location's contents and store 20 in it.

```
int x, *px;
x=10;
px = x;
```

Error: It should be `px = &x;`

```
int x=10, y=20, *px, *py;
px = &x, py = &y;
if(px<py)
printf("\n x is less than y");
else
printf("\n y is less than x");
```

Error: It should be `if(*px< *py)`

Points to Remember

- C was developed in the early 1970s by Dennis Ritchie at Bell Laboratories.
- Every word in a C program is either an identifier or a keyword. Identifiers are the names given to program elements such as variables and functions. Keywords are reserved words which cannot be used as identifiers.
- C provides four basic data types: `char`, `int`, `float`, and `double`.
- A variable is defined as a meaningful name given to a data storage location in computer memory.
- Standard library function `scanf()` is used to input data in a specified format.`printf()`function is used to output data of different types in a specified format.
- C supports different types of operators which can be classified into following categories: arithmetic, relational, equality, logical, unary, conditional, bitwise, assignment, comma, and sizeof operators.

- Modulus operator (%) can only be applied on integer operands, and not on float or double operands.
- Equality operators have lower precedence than relational operators.
- Like arithmetic expressions, logical expressions are evaluated from left to right.
- Both x++ and ++x increment the value of x, but in the former case, the value of x is returned before it is incremented. Whereas in the latter case, the value of x is returned after it is incremented.
- Conditional operator is also known as ternary operator as it takes three operands.
- Bitwise NOT or complement produces one's complement of a given binary number.
- Among all the operators, comma operator has the lowest precedence.
- `sizeof` is a unary operator used to calculate the size of data types. This operator can be applied to all data types.
- While type conversion is done implicitly, typecasting has to be done explicitly by the programmer. Typecasting is done when the value of one data type has to be converted into the value of another data type.
- C supports three types of control statements: decision control statements, iterative statements, and jump statements.
- In a `switch` statement, if the value of the variable does not match with any of the values of case statements, then default case is executed.
- Iterative statements are used to repeat the execution of a list of statements until the specified expression becomes false.
- The `break` statement is used to terminate the execution of the nearest enclosing loop in which it appears.
- When the compiler encounters a `continue` statement, then the rest of the statements in the loop are skipped and the control is unconditionally transferred to the loop-continuation portion of the nearest enclosing loop.
- A C program contains one or more functions, where each function is defined as a group of statements that perform a specific task.
- Every C program contains a `main()` function which is the starting point of the program. It is the function that is called by the operating system when the user runs the program.
- Function declaration statement identifies a function's name and the list of arguments that it accepts and the type of data it returns.
- Function definition, on the other hand, consists of a function header that identifies the function, followed by the body of the function containing the executable code for that function. When a function is defined, space is allocated for that function in the memory.
- The moment the compiler encounters a function call, the control jumps to the statements that are a part of the called function. After the called function is executed, the control is returned back to the calling function.
- Placing the function declaration statement prior to its use enables the compiler to make a check on the arguments used while calling that function.
- A function having `void` as its return type cannot return any value. Similarly, a function having `void` as its parameter list cannot accept any value.
- Call by value method passes values of the variables to the called function. Therefore, the called function uses a copy of the actual arguments to perform its intended task. This method is used when the function does not need to modify the values of the original variables in the calling function.
- In call by reference method, addresses of the variables are passed by the calling function to the called function. Hence, in this method, a function receives an implicit reference to the argument, rather than a copy of its value. This allows the function to modify the value of the variable and that change is reflected in the calling function as well.
- A pointer is a variable that contains the memory address of another variable.
- The & operator retrieves the address of the variable.
- We can 'dereference' a pointer, i.e., refer to the value of the variable to which it points by using unary * operator.
- Null pointer is a special pointer variable that does not point to any variable. This means that a null pointer does not point to any valid memory address. To declare a null pointer we may use the predefined constant NULL.
- A generic pointer is pointer variable that has `void` as its data type. The generic pointer can point to variables of any data type.
- To declare pointer to pointers, we need to add an asterisk (*) for each level of reference.