Teaching of Pointers in C

Version 2.0, 2003

Dr Achuthsankar S. Nair, University of Kerala

1 INTRODUCTION

Pointers are one of the most basic features of C which is both the strength and weakness of C. The full appreciation of the use of pointers will be possible only when you become an experienced C programmer. However, to start using pointers, all you need to know is a clear picture of the computer's memory. Let us study that first.

2 THE MAIN MEMORY

All the variables we have been using (and indeed, the program itself) reside in the memory when the program is executed. The organization of the memory is rather straightforward. It is a sequence of large number of memory locations, each of which has an address. Each memory location is capable of storing a small number (0 to 256), which we call a byte. A char data has 1 byte in size and hence needs one memory location of the memory. Both integer and float need 4 bytes each, or 4 locations(The size needed for a particular type varies with the platform in which the program is run. Even if an integer / float number is small, it will still occupy 4 locations. The following pictures in figure 1.2 represent these facts.

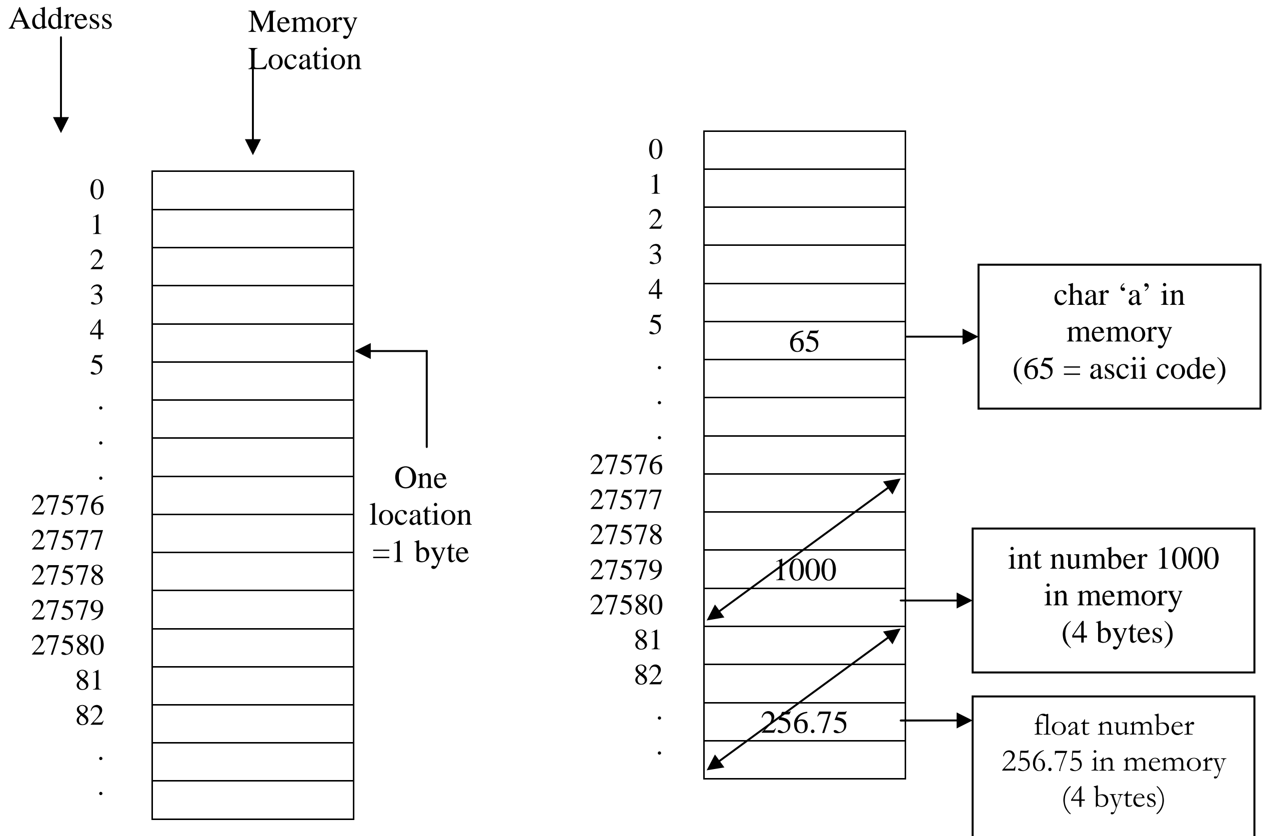

Fig 1. The Computer Memory

3 THE ADDRESS OF VARIABLES

All the variables that you declare in programs are allocated addresses in the memory. You can print that out using the & operator which you have already been using in scanf statements. Study the following program (1.):

Program 1

```
#include <stdio.h>

main( )

{

char x;

x = 'M';

printf("x = %d\n", x);

printf("Address of x = %c\n", &x);

}
```

If it were possible to 'peep' into the computer memory you would be able to see the following. (see fig 2)

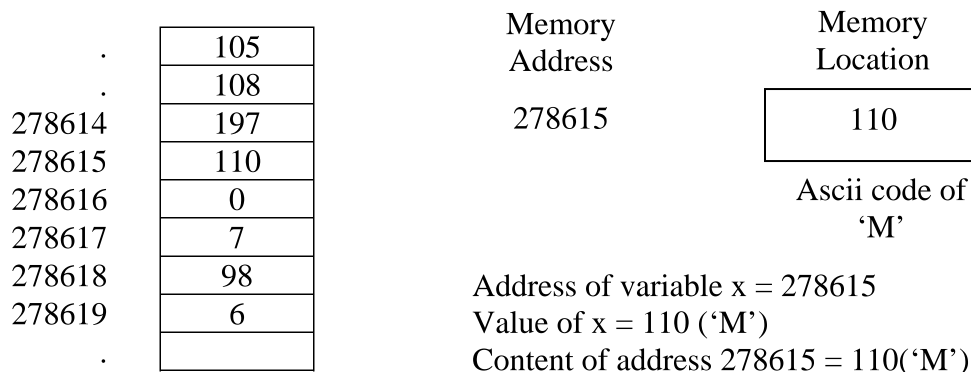

Fig 2

Consider program 2

Program 2

```
#include <stdio.h>
main()
{
int x;
x=1000;
printf("x=%d\n",x);
printf("Address of x = %d\n", &x);
}
```

If it were possible to 'peep' into computer memory you would be able to see the following:(see fig 3)

In this case, the value will be stored in 4 locations, not one, since integer requires 4 bytes to store. How 1000 is 'sliced' into 4 pieces, you need not bother now.

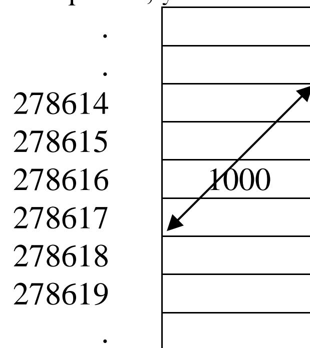

Fig 3

Here, what is the address of x? Actually it is 278614, 278615, 278616 and 278617. However in C we consider the address of C as 278614, the address of first of the 4 locations. This is an important thing to remember and crucial to the understanding of pointers.

The case of float data type is similar to the integer data type. Whenever variables are declared, some memory locations get allocated for them. It will be helpful to assume some addresses and draw the picture of the memory to answer the questions on pointers.

4. THE POINTER VARIABLE

In the previous section we printed out &x which is the address of x. Suppose we need to store the address in a variable. Then we need a special variable called the pointer. A pointer variable is one, which can store the addresses of another variable. Study the following program (3), which uses a pointers variable p.

Program 3

```
#include <stdio.h>
main( )
{
int x;
int *p;
x=1000;
p=&x;
printf(“x=%d\n”, x);
printf(“Address of x=%d\n”,p);
}
```

Address	Content
333333	
333334	1000
333335	
333336	
333337	
333338	
333339	333333
333340	
333341	

When you declare x, it gets allocated memory, say 4 locations starting from 333333. When pointer p is declared, it also gets allocated, say from 333337. When you assign x =1000; the value occupies the memory 333337 to 333336. Let us see what happens when you assign p=&x. As in any assignment statement, look at the R.H.S first. &x is the address of x which is 333333. This occupies the memory location starting from 333337 to 333340. Thus we can see that the pointer variable is like an int variable in some sense. The pointer declaration is given as follows:

int *p; ⟶ p is a pointer to an integer.

We can similarly declare pointer to char and float.

char *q; ⟶ q is a pointer to a char.

float *fp; ⟶ fp is a pointer to float.

What is the difference between a pointer to an integer, character & float? They are all storing memory addresses, isn't it? Well, recall that the address of the character is address of the single location in which the character is stored. But the address of integer / float only refer to the first address of the 4 locations in which integer / float is stored. That is why we need to declare a pointer as pointing to a certain data type. In general we can declare pointers as:

data type *pointer-name;

When assigning values to pointers, we have to take note of the data type to which the pointer points. For example:

i)
```
int x;
int *p;
p=&x;      ⇒Correct, p is a pointer to an int and &x is the address of an int
```

ii)
```
int x;
char *p;   ⇒ Not correct, p is a pointer to a char and &x is the address of an int
p=&x;
```

EXERCISE

Predict the output of the following program, or point out mistakes if any. Draw the picture of the memory for each and assume memory address shown consecutively.

1.
```
#include <stdio.h>
main()
{
int num;
int *inypoiny;
num=50;
intpoint=&num;
printf("The address of num is %d\n", &num);
printf("The address of intpoint %d\n", &intpoint);
}
```

Address	
223278	
79	
80	
81	
82	
83	
84	
85	
86	
87	

2.

```
#include <stdio.h>
  main( )
  {
  char grade;
  char *cp;
  grade='D';
  cp=&grade;
  printf("Grade is %c\n", grade);
  printf("Address of grade is %d\n", &grade);
  printf("cp is %d\n", cp);
  printf("Address of cp is %d\n", &cp);
  }
```

234555	
234556	
234557	
234558	
234559	
234560	
234561	
234562	
234563	
234564	

3.

```
#include <stdio.h>
  main( )
  {
  int a;
  char b;
  float c;
  int *ap;
  char *bp;
  float *cp;
  ap=&a;
  bp=&b;
  cp=&c;
  printf("The address of a, b and c are %d %d %d", ap, bp,cp);
  }
```

234555	
234556	
234557	
234558	
234559	
234560	
234561	
234562	
234563	
234564	

5 POINTER ARITHMETIC

Pointer arithmetic is an interesting aspect of pointers. We have already mentioned that pointers are very much like integers. C permits the use of some arithmetic operators on pointer variables. One can meaningfully apply addition and subtraction on pointers (+, -, ++ and --). The results are pleasant surprise. Study the following program (4).

Program 4

```
#include <stdio.h>
main()
{
char x;
char *p;
x='M';
p=&x;
printf("Pointer value =%d\n", p);
printf("Pointer plus one =%d\n", p+1);
 }
```

223455	77
56	
57	
58	
59	
60	
61	

An example output will be:

Pointer value = 234555
Pointer plus one = 234556

Of course, there is nothing surprising here.

Now, change the data type to integer and see program 5.

Program 5

```
#include <stdio.h>
main( )
{
        int x;
        int *p;
        x=1000;
        p=&x;
        printf("Pointer value = %d\n",p);
        printf("Pointer plus one =%d\n",p+1);
}
```

Address	Content
223455	
56	1000
57	
58	
59	
60	
61	

An example output will be:

Pointer value = 223455
Pointer plus one = 223439

223455+1=223439 ! That is pointer magic! How does C justify that? Well, 223455 is not just a number, C knows it is the address of an integer that takes 4 locations. So 223455, 223456, 223457 and 223458 are all together held by the integer. So C interprets +1 as next free location and gives the answer 223459.

Since float also takes 4 bytes to store, a pointer to float will also show the same effect. We can say that +1 is interpreted as follows by C:

char address + 1 ⟹ char address + 1
integer address + 1 ⟹ integer address + 4
float address + 1 ⟹ float address + 4

In general, address of any data type + 1 = address of the data type + (size of the data type in byte). This behavior of C is described as pointer arithmetic in C being scaled according to the data type.

6 POINTER DE-REFERENCING

So far we have only assigned values to pointers and tried incrementing them. There is another operation you can do with pointers, known as De-referencing. Before we proceed, be aware that '*' symbol appears in C language in 4 different situations with four different meanings. Three of these we have already seen

(i) Comments/*…*/

(ii) arithmetic operator for multiplication as in a*b

(iii) declaring pointer variables as in int *p.

(iv) De-referencing.

Now we will consider the fourth situation. Comment is easily recognized and so is multiplication. The rest of the two situations are related to pointers. When it appears in a declaration as in a declaration int *p, we just read it as p is a pointer to an integer. After the declaration, in the body of the program we can use the * with p as *p which is read as De-reference p. De-referencing can be explained as follows. Every pointer stores some address. *p means the value stored in that address. To understand *p, we could replace p with some assumed address. *(333375) means the value stored in location 333375. In this sense * works in a way exactly opposite to &.

&x = address of variable x.
*p = content of address given by p.

See the program 6:

Program 6

```
#include <stdio.h>
main()
{
    int x;
    int *p;
    x=100;
    p=&x;
    printf("x=%d\n",x);            x=100
    printf("p=%d\n",p);            p=275675
    printf("*p=%d\n",*p);          *p=100
}
```

Address	Content
223455	
223456	100
223457	
223458	
223459	
223460	
223461	275675
223462	
223463	

Before you decide the format string for printing *p, please check the data type that p is pointing to. Before we end this chapter, the final question. What is *(&x) in the above program? Remember, & and * are opposing operators. You should be able to guess now.

EXERCISE

1. Write the following pointer declarations.
   (a) p, a pointer to an integer
   (b) char p, a pointer to a character
   (c) fp, a pointer to a float
   (d) sp a pointer to struct student which has already, been declared. [Hint: Remember that once a structure is declared, they can be given the some treatment a sint, float, char]
2. Declare variables of type int, char, float and struct student and then assign their addresses to the respective pointers declared in Q1.
3. Draw the memory diagram for each of the above cases.
4. Predict the output of each of the following program (draw the memory diagram so that it will be easy to answer) where memory addresses are to be described; you can assume any 6-digit number. Assume numbers starting from 333333.

a)
```
int a;
int *integer_pointer;
a=222;
integer_pointer=&a;
printf("The value of a a %d\n", a);
printf("The address of a %d\n",&a);
printf("The address of integer_pointer %d\n", &integer_pointer);
printf("Star integer_pointer %d\n", *integer_pointer);
```

b)
```
for char
char a;
char *char_pointer;
a='b';
char_pointer=&a;
printf("The value of a %d\n", a);
printf("The address of a %d\n", &a);
printf("The address of char_pointer %d\n", &char_pointer);
printf("Star char_pointer %d\n", *char_pointer);
```

c)
```
for float
float a;
float *float_pointer;
a=22.25;
float_pointer=&a;
printf("The value of a %d\n", a);
printf("The address of a %d\n", &a);
printf("The address of float_pointer %d\n", &float_pointer);
printf("Star float_pointer %d\n", *float_pointer);
```

d)
```
int a, b
int *ip1, *ip2;
a=5;
b=6;
ip1=&a;
ip2=ip1;
printf("The value of a is %d\n", a);
printf("The value of b is %d\n", b);
printf("The address of a is %d\n",&a);
printf("The address of b is %d \n"&b);
printf("The address of ip1 is %d\n", &ip1);
printf("The address of ip2 is %d\n", &ip2);
printf("The value of ip1 is %d\n",ip1);
printf("The value of ip2 is %d\n", ip2);
printf("ip1 dereferenced %d\n",*ip1);
printf("ip2 dereferenced %d\n", *ip2);
```

e)
```
int i, j, *ip;
i=1;
ip=&i;
j=*ip;
```

```
*ip=0;
printf("The value of i %d\n", i);
printf("The value of j %d\n", j);
```

f)

```
int x, y;
int *ip1, *ip2;
y=1;
ip2=&y;
ip1=ip2;
x=*ip1+y;
printf("The value of x %d\n", x);
printf("The value of y %d\n",y);
```

© Achuthsankar S Nair, 2003. This article may be reused without alteration in any form, provided this notice is retained.