POINTERS IN C

PART II

Dear Friends,

Welcome back to the second session of Pointers in C. In this session we will discuss more features of pointers which help you to thoroughly understand the concept.

Array of pointers

Whenever addresses are stored as array elements, such an array is called an array of pointers. The format for the declaration of an array of pointers is

data type * array_name[size];

For example, the declaration float *a[10]; allocates the memory for 10 pointers, which are to be stored in the variables a[0], a[1], …., a[10]. These variables are not initialized. When the array elements are initialized by the following statements,

```
float f1=13.32, f2=14.23;
a[0] = &f1;
a[2] = &f2;
```

The values of f1 and f2 can also be obtained by the expression *a[0] and *a[2] respectively.

Consider the statements

```
float x[15][25];
float *y[15];
y[0]=x;
y[1]=x[1];
y[2]=x[2];
…….
……..
y[14]=x[14];
```

The first statement is a declaration of two dimensional array, the second one is a declaration of an array of pointers and the third one is an assignment of the address of the first element in the array of x to the first element y[0] of the pointer array y. Likewise, the starting address of the ith row is assigned to y[i]. Now the array x is also represented by the array y. Each element of the array y points to a 25 element array. Totally the memory locations for 375 (15*25) floats plus memory locations for 15 points are reserved. Thus , a 2D array may also be defined by a 1D array of pointers.
The size of the array of pointers in the declaration represents the number of rows in a 2D array.

Pointers and character arrays

Another important use of pointers is in handling of character arrays. Like the other arrays, the **array name** of the character array also gives the address of the zeroth element in that array. In the following statements,
Char *p;
P=”POINTER IN C”;

The address of ‘P’ is assigned to the pointer p. The second statement is actually initialization of pointer and not string copying. After this assignment it is possible to use p[0], p[1] so on.

Whenever more than one string are to be stored in an array the two-dimensional array is required to store each string in a row. Consider the following array of strings:

char cities [3][20];

It says that ‘cities’ is a table containing 3 names, each with a maximum length of 20 characters including null characters.

We know that rarely the individual strings will be of equal lengths. Therefore instead of making each row affixed number of characters, we can make it a pointer to a string of varying length. For example:

```
Char *cities[3] = {
                    "Hyderabad",
                    "Delhi",
                    "Kochi"
                };
```

Declares cities to be an array of three pointers to characters, each pointer pointing to a particular name as:

cities[0] ⟶ Hyderabad
cities[1] ⟶ Delhi
cities[2] ⟶ Kochi

It is also possible to use NUL character to find the end of the string. Fro example, while(*p!='\0') can be simply written as while(*p) because '\0' is equal to the value 0.

Different Types of Pointers

Using const with pointers

Using the **const** keyword in pointers we have to distinguish between making the pointer itself constant and making the value is pointed to constant.

Go through the following declarations;

const float *ptr; or **float const *ptr;** → ptr points to a constant float value.

That is the ptr points to a value , that must remain constant. The value of ptr itself can

be changed.

float *const ptr; → ptr is a const pointer

that is the pointer ptr cannot have its value changed. It must always point to the same address, but the pointed – to value can change.

const float * const ptr;

means both that ptr must always point to the same location and that the value stored at location must not change.

NULL pointers

We know that the pointer variable is a pointer to some other variable. There is one other value a pointer may have : NULL value, this is called as NULL pointer. A null pointer is a special pointer not pointing to any variable , array or any valid memory location.
The null pointer will hold null value and its content.

```
int *pt = NULL;

int *pt = 0; is also legal.
```

Void pointer

A void pointer is called as generic pointer which can point to any type of memory block. But to access the complete memory one has to specify casting explicitly. Consider the a program segment

```
#include<stdio.h>
main()
{
        Int a =1;
        Char c='a';
        Void *ptr;
        Ptr = &num;
        Printf("%d",*(int *) ptr);
        Ptr = &c;
        Printf("%c",*(char *) ptr);
}
```

Here the void pointer ptr is used as integer and character pointers suing explicit casting

method.

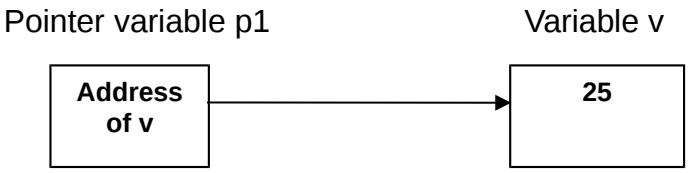
Pointer to pointer

Consider the statements,

int *p1, v=25;

p1=&v;

the pointer variable p1 has an address, which points to an int value 25 of the variable v, as in the figure shown on the screen.



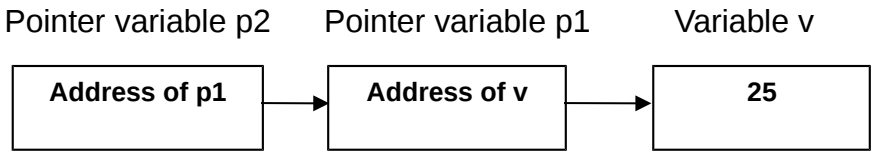
The value stored in the v can be obtained by using the expression *p1. If a pointer variable points to another pointer variable, then the situation may be imagined based on the statements

int *p1, **p2, v=25;

p1=&v;

p2=&p1;

as shown in the figure.



Such a situation is known as pointer to pointer. First the address of v is assigned to p1

and the address of p1 is assigned to p2. The expression *p1 gives the value of v and *p2 gives the value of p1, which is the address of the variable v. Hence the value of v can also be obtained by the expression **p2. Here two consecutive indirection operations are used. The expressions **p2, *p1 and v represent the same value 25.

Use of pointers in static and dynamic memory allocation

Let us consider the declarations,

char *cptr;
flaot *fptr;
double *dptr;
the compiler reserves the same amount of memory space for each pointer variable because each pointer variable stores only one address. The declaration reserves memory locations for the pointer variables, but it does not assign addresses to those pointer variables. So an appropriate address must be assigned to the pointer variable without fail before using it in the program. The declaration of pointer variable may include an initialiser, which should be an address. Such an assignment is known as initialization of pointer variables. Pointer variables may be initialized in the declaration itself or by using an assignment statement somewhere after the declaration in a program. There are two ways to obtain a memory address.

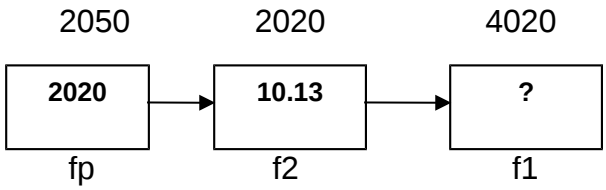
1. Static memory allocation
2. Dynamic memory allocation

In static memory allocation the address of a variable is assigned to a pointer variable. The compiler allocates or reserves the required memory spance for a declared variable. By using the address of the operator, the reserved address is obtained and this address is assigned to the pointer variable. Most of the declared variables have static memory (allocated memory space) within there scope. This

way of assigning pointer value is known as static memory allocation. The address of a variable having a specific data type must be assigned to a pointer variable declared to be the pointer to that specific data type. In this session we will discuss the use of & operator to get an address and assigning the address to a pointer variable . The declarations,

float f1, f2, *fp = &f2;
*f2=10.13

Initializes the pointer variable fp with the address of f2 as shown in the screen.



The address is assigned in the declaration itself, it is known as initialization of pointer variables.

It is important to note that the interpretation of the indirection operation * in the declaration statement is different from that of the operator * in the other places. The declaration

float f1,f2, *fp = & f2;

Informs the compiler that the expression *fp, f1 and f2 are float values. The syntax of the declaration of a pointer variable mimics the syntax of the expression that can appear while using it in order to represent its data object. The representations of the variables in the declaration are used as such to refer to the data objects. Thus, *fp also represents a float value like the variables f1 and f2. The indirection operator is not

applied to the variable in the declaration. But in the places other than declarations, the indirection operator is applied and the expression is evaluated. In the above declaration the pointer variable fp (not *fp) is initialized with the address of the variable f2. The value 10.13 is assigned to *fp using the assignment statement *fp=10.13. These statements may be equivalently written as

```
float f1,f2,*fp;
fp=&f2;
*fp=10.13;
```

To assign the address of f2 to fp. The declaration

```
float f1, *fp = &f2, f2;
```

Does not initialize fp because the variable f2 is used before its declaration. Since C does not have look-ahead capability, this type of initialization is not possible. In the following statements,

```
float f1, *fp;
Int I;
fp=&I;  /* invalid assignment */
```

Dynamic memory allocation

It is a process by which the required memory address is obtained without an explicit declaration. The required memory space is obtained by using the memory allocation functions like malloc() and calloc(). Memory allocation functions return a pointer value to a block of memory of required size. This returned pointer value is assigned to a pointer variable. Thus the required memory space is dynamically created and assigned. This method of memory allocation is known as dynamic memory allocation. The storage space allocated dynamically has no name and hence its contents can be accessed only through a pointer.

First the number of memory locations required for holding a data object must be known . C provides an operator sizeof to find the required space to store the data object of a

particular data type. The format of a sizeof operator is

Sizeof(data-type or variable);

For example sizeof(int) retures the size of memory to hold an int data type. The header file stdlib.h should be included at the beginning of the program while using the memory allocation functions.

The function malloc() obtaines the number of bytes mentioned as its argument and returns a generic pointer (pointer to void that is void *) to the memory obtained. For example, in the statements,

```
int *ip;
ip=(int*) malloc(sizeof(int));
*in=20;
```

OR

```
Int *ip= (int*)malloc(sizeof(int));
*ip=20;
```

The pointer variable ip obtains exactly the right amount of memory to hold a data object. The function malloc() uses one argument only which represents the number of bytes to be reserved for storage. The memory allocated by this function contains garbage values. The cast operator is used to convert the generic pointer (void *) returned by the function malloc() to the concerned data type. Observe the casting (int *) in the pointer assignment statements that convert the generic pointer to an int pointer. The declaration int *p=(int *)malloc(4*sizeof(int));

Reserve a block of memory containing 8 (4*2) memory locations for holding four in t values.

The function calloc () is similar to mallaoc() but uses two arguments. The first argument represents the number of data objects for which the memory is to be allocated and the

second argument represents the size of each data object. The memory allocated by this function contains the storage initialized to zero. For example,

```
Flaot *ip;
Ip=(flaot *) calloc(10,sizeof(float));
```

Obtain the space for 10 float data objects (having the values 0). The function calloc() is quite suitable for obtaining space for arrays.

The function realloc() is capable of increasing or decreasing the space that has been allocated previously. For example, consider the statements,

```
Float *fp;
Int n=10;
Fp=(float *)malloc(5*sizeof(float) );
Fp=realloc(fp,n);
```

The first argument in realloc() is a pointer to a block of memeory of which the size is to be altered. The second argument n specifies the new size. If the allocation is successful, the returned value is again the pointer to the first byte of the allocated memory retaining the old contents. If n is greater than the old size and if sufficient additional space is not available subsequent to the old region of the function realloc() may create a new region. Also the old data are moved to the new region.

The function malloc(), calloc() and realloc() obtain memory space dynamically. They call upon the operating system to obtain more blocks of memory as needed. The main advantage of using these memory allocation functions is getting the amount of memory as per the requirements without wastage of memory. When the requirement of memory is not known in advance, such functions will be helpful to obtain memory space as and when required. All these functions return the generic pointer. By using a cast operator, the appropriate type of pointer is obtained. If the required size of memory is not available in the computer memory, NULL is returned by all these functions, The NULL value may be used for verifying the successful creation of memory space as given in the statements,

```
Int *p;
```

```
If((p=(int*) malloc(sizeof(int) * 20))==NULL)
{
Printf("Unsuccessfuk emmory allocation \n");
Exit(1);
}
```

The memory obtained by malloc(), calloc() and realloc() functions can be freed when that space is not required for any storage. The function free(0 does this job. The function call

```
free (ptr);
```

makes the space, which is obtained by malloc() or calloc() or realloc() and pointed by the pointer variable ptr. The function call realloc(ptr,0) is equivalent to the function call free(ptr).

When the pointer is used we must systematically follow certain steps

1. Declare the pointer variable
2. Initialize the pointer variable with an appropriate address.
3. If the content of the address stored in the pointer is required, the indirection operator * is used to obtain value.

Pointer to a Function

We have already discussed the use of pointers in the call by reference method of parameter passing in the functions. By passing the pointers, the modified contents are also available in the calling function. Without pointers, this effect is not possible in a program. Function can also be passed as the arguments in another function by using pointers to those functions. A function may also return a pointer to a data type.

This section can be appreciated only by the advanced users of C. A function is not a variable, but it has an address. The function name itself represents the address of that function. Hence it is possible to define a pointer to a function. Functions are stored in memory similar to a data. A function can be assigned, placed in the arrays, passed as an argument to other functions and returned by a function by means of a pointer to a function.

The declaration of a pointer to a function has the format

data_type (*function_name) (datatype_1, datatype_2, ……, datatype_n); and function call has the format

(*function_name)(arg_1,arg_2,……..,arg_n)

The parentheses in (*function_name) are necessary, since in *function_name() the parentheses have higher precedence than * and they are evaluated from left to right.

If the parentheses are omitted, then it becomes a function returning a pointer. The general format for a function returning a pointer is

data_type * function_name(arg_1,arg_2,……., arg_n);

The function call of a function returning a pointer is similar to the ordinary function call as given by

*function_name(arg_1, arg_2,……, arg_n);

Thus the pointer to a function must be declared carefully to avoid the confusion between a function returning a pointer and a pointer to a function, For example,

```
double (*fp)();  /* fp is a pointer to a function */
double *fun();  /* fun is a function returning a pointer */
```

Command line arguments

The function main() in C can also pass arguments or parameters like the other functions. It has two arguments argc (for argument count) and argv (for argument vector). It may have one of the following two forms

```
Main( int argc, char *argv[])
```

Or

Main(int argc, char ** argv)

Where

argv represents the number of arguments

argv is a pointer to an array of strings or pointer to pointer to character.

The argument argv is used to pass strings to the programs. Hence, the arguments argc and argv are called as program parameters. After successful compilation, the program is executed by using an execution command. The execution command depends on the system running the program. It is possible to pass the value of program parameter argv through an execution command only. Actually, the execution command is a command line. The value of the program parameter argv is given in the command line. The value of argv is automatically counted by the compiler when main() is invoked. Since the values are available and obtained from the command line, they are also known as command line arguments.

The values of the arguments are generally passed through the actual arguments when the function is invoked. Since main() is the first function invoked during execution, the values are obtained from the command line itself.

It can be observed that the execution of pointer version programs are faster than the programs without using pointers. There are instances, where pointers play important roles and certain operations and effects are possible only with the help of pointers.

Summary

Let us now summarize the points that we discussed in the session.

1. We have understood that an array of pointers in an array that can hold the addresses of other variables.
2. The execution of the declaration statement results in allocating storage spaces only and the data objects are not assigned automatically. Hence after the declaration of

the pointer variables, they must be initialized. A pointer variable may be initialized by using static or dynamic memory allocation. In static memory allocation, the space reserved by the compiler is assigned to a pointer variable. Dynamic memory allocation is obtained by using the built-in functions like malloc() and calloc(). A dynamically created memory may be freed using the function free().

3. It is also possible to define a pointer to a function, for manipulating the function as an ordinary variable.
4. It is possible to pass string variables to the function main() using the standard parameters argc and *argv[] in it. A pointer is a valid address, which is stored in a pointer variable.

Please try to completely solve this **assignment.**

1. Functions and pointers work together in C. Explain.
2. What does the sizeof return to an array name?
3. What is the purpose of the command line argument?
4. How can a function be passed as an argument?
5. Write a program to merge two unsigned numeric arrays using pointers.

Here is some books for your **Reference**

1. B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.
2. Yashavant Kanetkar; Let us C, BPB Publications, New Delhi.
3. Greg Perry, Absolute Beginners' guide to C, SAMS publishing, April 1994.

Hope you start enjoying the lessons of C and interested in writing C programs. There are many more other features of C are remaining to explore. So let us wait for the coming sessions and till then bye._