Pointers: Introduction to pointers, declaring pointer variables, Types of pointers, Passing arguments to functions using pointers

UNDERSTANDING THE COMPUTER'S MEMORY

- Every computer has a primary memory. All data and programs need to be placed in the primary memory for execution.
- The primary memory or RAM is a collection of memory locations and each location has a specific address. Generally, the computer has 3 areas of memory each of which is used for a specific task. These areas of memory include – Stack, Heap and Global memory.
- **Stack:** A fixed size of stack is allocated by the system and is filled as needed from the bottom to the top, one element at a time. These elements can be removed from top to the bottom by removing one element at a time. When the program has used the variables or data stored in the stack, it can be discarded to enable the stack to be used for other programs to store its data.
- **Heap:** It is a contiguous block of memory that is available for use by the program when the need arises. A fixed size heap is allocated by the system in a random fashion. The addresses of the memory locations in the heap that are not currently allocated to the program for use are stored in a free list. When the program requests a block of memory, the dynamic allocation technique takes a block from the heap and assigns it to the block, it returns the memory block to heap and the location of the memory locations in that block is added to the free list.
- **Global Memory:** The block of code that is the main() program is stored in the global memory. The memory in the area is allocated randomly to store the code of different functions in the program in such a way that one function is not contiguous to another function. Besides, the function code, all global variables declared in the program are stored in the global memory area.

Introduction to Pointers:

- A pointer is a variable that holds the address of another variable. The pointer variable contains only the address of the referenced variable not the value stored at that address.
- Some of the advantages of using pointers are as follows

1. Pointers are more efficient in handling arrays and data tables
2. Pointers are used with function to return multiple values
3. Pointers allow C to support dynamic memory management

4. Pointers provide an efficient tool for manipulating dynamic data structures such as structures, linked list, stacks, queues, trees etc
5. Pointers reduce length and complexity of program
6. Pointers reduce program execution time and saves data storage space in memory

- Disadvantages of pointers

1. One of the disadvantage of using pointer is that program may crash if sufficient memory is not available at run time to store pointers.

- Pointer uses two basic operators

1. **The address operator (&):** It gives address of an object

2. **The indirection operator (*):** It is used to accesses the object the pointer points to.

Declaring a Pointer:

- A pointer provides access to a variable by using the address of that variable.
- The general syntax:

data_type *ptr_name;

Data type: It specifies the type of the pointer variable that you want to declare like (int, float, char, double or void)

*** (Asterisk):** tells the compiler that you are creating a pointer variable and ptr_name specifies the name of the pointer variable.

Example:

int *p; // declares a pointer variable p of integer type

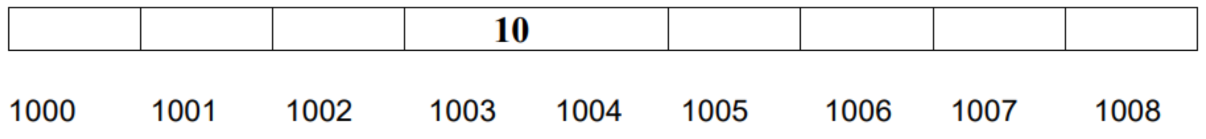
float *temp; // declares a pointer variable temp of float data type

Now consider, example

int x = 10;

int *ptr;

ptr = &x;



- Now, since x is an integer variable, it will be allocated 2 bytes.
- Assuming that the complier assigns it memory locations 1003 and 1004, we say the value of x=10 and address of x is equal to 1003. We can dereference a pointer, i.ie, refer to the value of the variable to which it points, by using '*' operator as in *ptr.

Example: Program of usage of pointer variable

```
#include<stdio.h>

void main()

{

        int x = 99;

        int *ptr; // Declare a pointer

        ptr = &x; // Assign the value to pointer

        printf("Value of ptr = %d\n", *ptr);

        printf("Address of ptr =%p\n", ptr);

}
```

Output:

```
Value of ptr = 99

Address of ptr =0x7ffdba4e00e4
```

Also note that it is not necessary that the pointer variable will point to the same variable throughout the program. It can point to any variable as long as the data type of the pointer variable it points to.

The following code illustrates this concept

```
# include<stdio.h>

void main()

{

    int a=3, b=5;

    int *pnum;
    pnum = &a;

    printf("\n a = %d\n",*pnum);

    pnum = &b;

    printf("b = %d\n",*pnum);

}
```

Output:

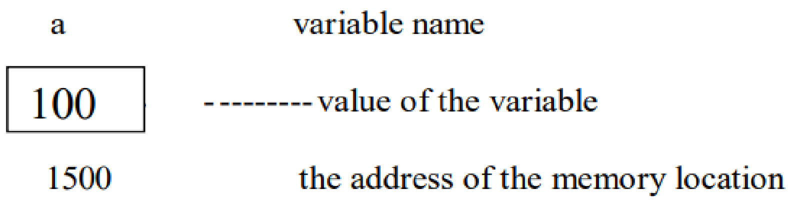
```
a = 3

b = 5
```

Using the address of (&) operator:

- A computer uses memory to store the instructions of different programs and the values of different variables.
- Since memory is a sequential collection of storage cells, each cell has an address.
- When you declare a variable, the operating system allocates memory according to the size of the data type of that variable.
- In this memory location, the value of the variable is stored.

Example : int a=100;

This statement request the operating system to allocate two bytes of space in memory and stores 100 in that location.


a
variable name
100
---------value of the variable
1500
the address of the memory location

By using the address of (&) operator, you can determine the address of a variable.

Pointer Expressions and Pointer Arithmetic

- Pointer variables can also be used in expressions. For ex,

```
        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;
```

- We can add integers to or subtract integers from pointers as well as to subtract one pointer from the other.
- We can compare pointers by using relational operators in the expressions. For example p1 > p2 , p1==p2 and p1!=p2 are all valid in C.
- When using pointers, unary increment (++) and decrement (--) operators have greater precedence than the dereference operator (*). Therefore, the expression

  *ptr++ is equivalent to *(ptr++). So the expression will increase the value of ptr so that it now points to the next element.
- In order to increment the value of the variable whose address is stored in ptr, write (*ptr)++.

Summarize the rules of pointers:

- A pointer variable can be assigned the address of another variable (of the same type).
- A pointer variable can be assigned the value of another pointer variable (of the same type).A pointer variable can be initialized with a null value.
- Prefix or postfix increment and decrement operators can be applied on a pointer variable.
- An integer value can be added or subtracted from a pointer variable.

- A pointer variable can be compared with another pointer variable of the same type using relational operators.
- A pointer variable cannot be multiplied by a constant.
- A pointer variable cannot be added to another pointer variable.

WAP to add 2 floating point numbers using pointers.

```
# include<stdio.h>

void main()

{

    float num1,num2,sum=0.0;

    float *pnum1=&num1,*pnum2=&num2,*psum=&sum;

    printf("Enter 2 numbers: ");

    scanf("%f,%f",pnum1,pnum2);

    *psum=*pnum1+*pnum2;

    printf("%f + %f = %f",*pnum1,*pnum2,*psum);

}
```

Output:

```
Enter 2 numbers: 3.2,2.3

3.200000 + 2.300000 = 5.500000
```

WAP to test whether a number is positive, negative or equal to zero

```
#include<stdio.h>

void main()

{

    int num,*pnum=&num;

    printf("Enter any number: ");

    scanf("%d",pnum);
    if(*pnum>0)
        printf("The number is positive");
    else if(*pnum<0)
        printf("The number is negative");
    else
        printf("The number is equal");
}
```

Output:

```
Enter any number: 2
The number is positive
```

WAP to print all even numbers from m to n.

```
#include<stdio.h>
void main()
{
    int m,*pm=&m;
    int n,*pn=&n;
    printf("Enter the starting and ending limit: ");
    scanf("%d,%d",pm,pn);
    while(*pm<=*pn)
    {
        if(*pm%2==0)
            printf("%d is even\n",*pm);
        (*pm)++;
    }
}
```

Output:

```
Enter the starting and ending limit: 0,10
0 is even
2 is even
4 is even
6 is even
8 is even
10 is even
```

Null Pointers

- A null pointer which is a special pointer value that is known not to point anywhere. 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,

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

- You can always check whether a given pointer variable stores address of some variable or contains a null by writing,

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

- Null pointers are used in situations if one of the pointers in the program points somewhere some of the time but not all of the time.
- In such situations it is always better to set it to a null pointer when it doesn't point anywhere valid, and to test to see if it's a null pointer before using it.

Generic Pointers:

- A generic pointer is pointer variable that has void as its data type.

- This is also called void pointer.
- The generic pointer, can be pointed at variables of any data type.
- It is declared by writing

  void *ptr;

- You need to cast a void pointer to another kind of pointer before using it.
- Generic pointers are used when a pointer has to point to data of different types at different times.

Example:

```
#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
```

- **gp:** It is a pointer variable.

- **(int*)gp:** This part involves type casting. It's casting the pointer gp to a pointer to an integer (int*).
- ***(int*)gp:** Finally, it dereferences the casted pointer, retrieving the value stored at the memory location pointed to by gp, but interpreting it as an integer.

PASSING ARGUMENTS TO FUNCTION USING POINTERS

- The calling function sends the addresses of the variables and the called function must declare those incoming arguments as pointers.
- In order to modify the variables sent by the caller, the called function must dereference the pointers that were passed to it.
- Thus, passing pointers to a function avoid the overhead of copying data from one function to another.

WAP to add 2 integers using functions.

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

Output:

```
Enter two numbers : 2,3
Total = 5
```

WAP to find the largest of 3 integers using functions.

```
#include<stdio.h>
void large( int *a, int *b, int *c, int *lar);
int main()
{
        int num1, num2, num3, big;
        printf("\n Enter 3 numbers : ");
        scanf("%d,%d,%d", &num1, &num2, &num3);
        large(&num1, &num2, &num3, &big);
        return 0;
}
void large( int *a, int *b, int *c, int *lar)
{
        if(*a>*b && *a>*c)
                *lar=*a;
        else if(*b>*a && *b>*c)
                *lar=*b;
        else
                *lar=*c;
        printf("The largest number is %d",*lar);
}
```

Output:

```
Enter 3 numbers : 2,5,6
The largest number is 6
```

Application of Pointers

- **Arrays and Strings:**

  Pointers can be used to traverse and manipulate arrays and strings more efficiently.

- **Function Pointers:**

  Pointers to functions allow you to create arrays of functions, pass functions as arguments to other functions, or return functions from functions.

- **Structures:**

  Pointers can be used to work with structures more efficiently, especially when dealing with large structures.

- **File Handling:**

  Pointers are used in file handling operations, enabling efficient reading and writing of data to files.

- **Efficient Parameter Passing:**

  Passing pointers to functions instead of passing large data structures can be more efficient in terms of both time and space.

- **Dynamic Data Structures:**

  Pointers are crucial in implementing dynamic data structures such as trees and graphs.

- **Callback Functions:**

  Pointers to functions are often used for implementing callback mechanisms, where a function can be passed as an argument to another function.