Lesson 7: Pointers

7.1 Introduction to Pointers

A pointer is an object that contains the memory address of another object. It is one of the most useful features of c++ but can also be troublesome if misused. It can be used to:

1. Support linked lists
2. Support dynamic memory allocation
3. Alter the contents of an argument.

The general form of a pointer variable declaration is:

```
type *var_name;
```

Where type is the pointer's base type which determines the type of data the pointer will be pointing to.

Illustration:

```
int *p;//declares p to be a pointer to an integer.
float *f;// declares f to be a pointer to an float.
```

In general preceding a variable name with an asterisk causes it to become a pointer. The * obtains the value stored at the address that it precedes and is called the dereference operator. To obtain the address of an object we use the reference operator (&) also called an address operator as illustrated in the example below.

Example1:

```
#include<iostream.h>
int main()
{
        int num;
        int *p;
        int val;
        num=62;
        p=&num; //get address of num
        val=*p; //get value at that address
        cout<<"The value stored at address\t"<<p<<"\t is\t"<<val<<endl;
return 0;
}
```

The output may be similar to this one:

The value stored at address Ox3fffd16 is 62

It is important to initialize pointers when they are declared. This can be illustrated using the declaration statement: int *p; .When a pointer is declared like this, it only allocates memory for the pointer itself. The value of the pointer will be some memory address, but the memory at that address is not yet allocated. This means that storage could be in use by some other variable. In this case p is not initialized, that is, it is not pointing to any allocated memory. Any attempt to access the memory to which it points will be an error
*p=62; //Error: no storage has been allocated for *p
This is a common problem created when using pointers. To avoid it, initialize pointers when they are declared for example:

```
int num=62; //num contains the value 62
int *p=&num; // p contains the address of num
cout<<*p;
```

In this case, accessing *p is no problem because the memory needed to store the integer 62 was automatically allocated when num was declared; p points to the same allocated memory.
It is important to ensure that pointer variables always point to the correct data type. Although one may use a cast to assign one type of pointer to another, it is not a good idea since it may result to garbage output.

7.2 Assigning values through a pointer.

You can use a pointer to assign a value to the location pointed to by the pointer as shown in the example below:
Example 2:

```
#include<iostream.h>
int main()
{
        int *p,num;
        p=&num;
        *p=100;//assign num the value of 100 through p
        cout<<num<<' ';
        (*p)++;//increament num through p
        cout<<num<<' ';
        (*p)--;//decreament num through p
        cout<<num<<"\n";
return 0;
}
```

The output will be 100 101 100
Note that in the statement (*p) ++; the parentheses is necessary because the * operator has lower precedence than ++ operator.

7.3 Pointer Arithmetic

There are only four operators that can be used in pointers: ++, --, +, - .
Below is an illustration of how the operators are used and their effects:
Let P1 be an integer pointer with a current value of 2000.

```
P1++;
```

This expression increases the value of p1 to 2004 and not 2001. This is because each time a pointer is incremented it will point to the next memory location of its base type. Since an integer is 4 bytes long (32 bits) then the next location is 2004 and not 2001.
Q if p1 was a character pointer, what would be its new value after the increment.
P1--;
Decreases the value of P1 to 1996 for the same reason explained above.
P1=P1+9;
This expression makes P1 point to the ninth element of P1's base type, beyond the one to which it is currently pointing. Note that only integers can be added or subtracted (P1=P1-9 ;).
You may subtract one pointer from another provided they are of the same base type but you can not add pointers. You may also compare pointers pointing to elements within the same array using relational operators such as = =, < and >.

7.4 Pointers and Arrays

Arrays and pointers have the following things in common.

1. The identifier of an array is equivalent to the address of its first element that it points. For example: consider the declarations below;
   ```
   int numbers [20];
   int *p;
   ```
   The following allocation would be valid
   ```
   p=numbers;
   ```
   At this point p and numbers are equivalent and they have the same properties. The only difference is that we can assign another value to the pointer p whereas numbers will always point to the first of the 20 integer numbers with which it was defined. Thus, unlike p which is a variable pointer, number, like all arrays is a constant pointer. This explains why the following allocation is not valid.
   ```
   numbers = p;   /*not valid because numbers is a constant pointer and no
                    values can be assigned to constant identifiers*/
   ```
2. The bracket operator (also known as offset operators) used to specify the index of an array is equivalent to adding the number within the brackets to the address of a pointer.
   Illustration:

   ```
   a [5] = 0;
   *(a+5) = 0;
   ```
   } are equivalent and valid if "a" is a pointer or an array.

   This is demonstrated in the examples below:

Example 3.1

```
// reverse case using array indexing
#include<iostream.h>
#include<cctype>

int main()
{
```

```
        int i;
        char str[30]="three is equal TO THREE";

        cout<<"Original String:\t"<<str<<"\n";
        for(i=0;str[i];i++)
        {
            if(isupper(str[i]))
                str[i]=tolower(str[i]);
            else if(islower(str[i]))
                str[i]=toupper(str[i]);

        }
        cout<<"Inverted case string:\t"<<str<<"\n";
        return 0;
    }
```

The output is:

Original String: three is equal TO THREE
Inverted case string: THREE IS EQUAL to three

Note:
The program uses the isupper(),islower(),tolower() and toupper() library functions from the cctype header file. Below is a description of how these functions work:

function	**explanation**
isupper()	Returns true when its argument is an uppercase letter
islower()	Returns true when its argument is a lowercase letter
tolower()	Returns the lowercase version of a letter if the letter is in uppercase
toupper()	Returns the uppercase version of a letter if the letter is in lowercase

The loop iterates until the null character is indexed. Since a null is zero (false) the loop stops.
The same program is rewritten below using pointer arithmetic.

Example 3.2

```
    //reverse case using pointer arithmetic
    #include<iostream.h>
    #include<cctype>
    int main()
    {
        char *p;
        char str[30]="three is equal TO THREE";

        cout<<"original String:\t"<<str<<"\n";
        p=str;//assing p the address of the strart of the array
        while(*p)
        {
            if(isupper(*p))
                *p=tolower(*p);
            else if(islower(*p))
```

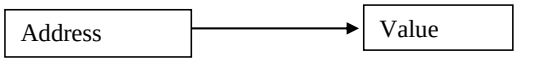
```
                *p=toupper(*p);
            p++;

        }
        cout<<"Inverted case string:\t"<<str<<"\n";
        return 0;
    }
```

Though the output is the above example is the same as in Example 3.1, the two programs are not equivalent in performance. The second method (using pointers) is more efficient and therefore recommended.

7.5 Pointers to Pointers

Consider the figure below:

1. Pointer Variable



2. Pointer Pointer Variable

The first case shows the normal pointer. In this case the value of a pointer is the address of a value.

The second case shows a pointer to a pointer. The first pointer contains the address of the second pointer, which points to the location that contains the desired value. This is a form of multiple indirection or a chain of pointers. The chain may be extended to whatever extent desired.

A variable that is a pointer to a pointer must be declared as such.This is done by placing an additional asterisk in front of its name.For example: ***int **balance***; is a pointer to an int pointer.

Example 4

```
#include<iostream.h>
int main()
{
    int x,*p,**q;
    x=10;
    p=&x;//assing p address of x
    q=&p;//assing q the address of p

        cout<<"the address of x is\t"<<p<<"\n";
        cout<<"the address of p is\t"<<*q<<"\n";
        cout<<"the value of x as referenced by q is \t"<<**q<<"\n";
return 0;
}
```

7.6 Arrays of Pointers

Pointers can be arrayed like any other data type. To declare pi as an array of 10 integer pointers we use the statement:

```
int *pi[10];
```

To assign the address of an integer variable called Val to the third element of the pointer array we write:

```
pi[2]=&Val;
```

The value of Val can be accessed using the statement:

```
*pi [2];
```

Arrays of pointers can be initialized. The example below demonstrates the most common use of pointer arrays, that is, to hold pointers to strings

Example 5

```
#include <iostream.h>
#include <cstring>
int main()
{
        char *dictionary[][2]={
                "pen","A writing instrument ",
                "keyboard","A computer input device",
                "c++","A programming language"
                "",""
        };
        char word[30]; // an array of 30 .
        int i;
        cout<<"Enter word\t";
        cin>> word;
        for(i=0;*dictionary[i][0];i++)
        {
        //strcmp(): member of the cstring header file. Compares two strings.
                if(!strcmp(dictionary[i][0],word))
                {
                        cout<<dictionary[i][1]<<"\n";
                        break;
                }
        }
        if(!*dictionary[i][0])
                cout<<word<<"\tnot Found\n";
return 0;
}
```

In the above program, each word entered by the user is compared to the words in the dictionary. If a match is found the meaning is displayed and the iteration stopped by the break statement. Program control goes to the statement immediately following the for loop. Otherwise, the for loop runs until the first character in a string is null. This explains why the dictionary ends with two null strings. C++ will store all the string constants in a string table associated with this program. The array stores pointers to the strings.

7.7 The new and delete operators.

The **new** operator

The new operator is used to request for dynamic memory allocation. It allocates memory and returns a pointer to the start of it. It may be used to assign memory to a single element or to several elements depending on how it is used. Below is an illustration of its use.

1. Pointer =new type; // as in int *p=new int;
2. Pointer= new type[elements]; // as in int *p=new int[5];

Statement 1 assigns memory to contain one single element of type int. Statement 2 assigns a block [array] of memory with space for 5 elements of type int and returns a pointer to the beginning of that memory block.

The above initialization statements ensure that memory is allocated at run time as compared to the normal declaration statements for arrays and other variables where memory would be allocated at compile time.

The **delete** operator

The delete operator reverses the action of the new operator by returning allocated memory to the store. It should only be applied to pointers that have been allocated explicitly by the new operator. Below is an illustration of its use:

```
float q=new float ;
*q=3.14;
delete q;
```

Dynamic Arrays

Below are two ways of defining an array.

1. float a [20];
2. float *q=new float [20];

The first method defines a static array that is created at compile time while the second method defines a dynamic array that is created at run time.

Commented [E1]: Memory allocate only when its declaration excutes.

Example: 6

```
// using the get function to create a dynamic array.
#include <iostream.h>
void main()
{
        double* a;// a is an unallocated pointer
        int n;

for(int k=1;k<=2;k++)
{
        cout<<"Enter number of items:";
        cin>> n;
        a=new double[n];// a is an array of n doubles
        cout<<"Enter "<<n<<" items,one per line:\n";
        for(int i=1;i<=n;i++)
        {
                cout<<"\t"<<i<<":";
```

```
                cin>> a[i];
                cout<<endl;
        }
        for (int j=1;j<=n;j++)
        {
                cout<< a[j]<<" ";
        }
        cout<<endl;
}

if(k==2)
delete []a; //a is an unallocated pointer


}
```

7.8 Exercise

1. Write a c++ program to find the sum of all even numbers in the array shown below:
   ```
   int Value[]={0,1,2,3,4,5,6,7,8,9,10};
   ```
2. Define a pointer and list three of its uses.
3. Explain the use the reference and dereference operators with respect to pointers.
4. Write declaration statements for the following:
   a. A pointer to a double
   b. A two dimensional array of type integer.
   c. A pointer to an integer pointer.
5. List four arithmetic operators that can be used in pointers.
6. Given that p is an integer pointer with a current value of 1000, explain the effect of the statements below;
   a. P++;
   b. p- -;
   c. p=p+1;
   d. p=p-1;

7. Write a program that reverses the case in the sentence “an ARRAY is JUST a CONSTANT pointer” using:
   a. Array indexing
   b. Pointer arithmetic
8. Explain the use of the new and the delete operators in creating dynamic memory.
9. Show how the following array definitions are done:
   a. A dynamic array of 20 floats
   b. A static array of 20 floats