Chapter 11 - Pointers

POINTERS

Introduction:

- Pointers are a powerful concept in C++ and have the following advantages.
  1. It is possible to write efficient programs.
  2. Memory is utilized properly.
  3. Dynamically allocate and de-allocate memory.

Memory Utilization of Pointer:

- Memory is organized as an array of bytes. A byte is basic storage and accessible unit in memory.
- Each byte is identifiable by a unique number called address.
- We know that variables are declared before they are used in a program. Declaration of a variable tells the compiler to perform the following.
  - Allocate a location in memory. The number of location depends on data type.
  - Establish relation between address of the location and the name of the variable.
- Consider the declaration, `int num;`
- This declaration tells the compiler to reserve a location in memory. We know that size of int type is two bytes. So the location would be two bytes wide.

<table>
<thead>
<tr>
<th>Address</th>
<th>Num</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>15</td>
</tr>
<tr>
<td>101</td>
<td></td>
</tr>
</tbody>
</table>

- In the figure, num is the variable that stores the value 15 and address of num is 100. The address of a variable is also an unsigned integer number. It can be retrieved and stored in another variable.

Pointer:

- A pointer is a variable that holds a memory address of another variable.
- The pointer has the following advantages.
  - Pointers save memory space.
  - Dynamically allocate and de-allocate memory.
  - Easy to deal with hardware components.
  - Establishes communication between program and data.
  - Pointers are used for file handling.
  - Pointers are used to create complex data structures such as linked list, stacks, queues trees.
and graphs.

**Pointer Declaration:**

- Pointers are also variables and hence, they must be defined in a program like any other variable.
- The general syntax of pointer declaration is given below.
  
  Syntax: `Data_Type *Ptr_Variablename;

- Where,
  - `Data_Type` is any valid data type supported by C++ or any user defined type.
  - `Ptr_Variablename` is the name of the pointer variable. The presence of `*` indicates that it is a pointer variable.
- Defining pointer variables:
  - `int *iptr;` `iptr` is declared to be a pointer variable of int type.
  - `float *fptr;` `fptr` is declared to be a pointer variable of float type.
  - `char *cptr;` `cptr` is declared to be a pointer variable of character type.

**Pointer Initialization:**

- Once we declare a pointer variable, we must make it to point to something.
- We can do this by assigning or initializing to the pointer the address of the variable you want to point to as in: `iptr = &num;`
- The `&` is the address operator and it represents address of the variable.
- Example: A program to display the content of `num` and the address of the variable `num` using a pointer variable.

```
#include<iostream.h>
void main()
{
  int num; // normal integer variable
  int *iptr; // Pointer declaration, pointing to integer data
  num = 574; // assign value to num
  iptr = &num // assign address of num to int pointer
  cout<<"Value of num is ":"<<num<<endl;
  cout<<"Address of num is ":"<<iptr<<endl;
}
```

**The address of operator (&):**

- `&` is a unary operator that returns *the memory address of its operand*.
- For example, if `var` is an integer variable, then `&var` is its address.
- We should read `&` operator as “the address-of” which means `&var` will be read as “the address of
var”.

Example:

```c
int num = 25;
int *iptr;
iptr = &num;  //The address of operator &
```

(Pointer Operator or Indirection Operator (*):

- The second operator is indirection operator ‘*’, and it is the complement of ‘&’.
- It is a unary operator that returns the value of the variable located at the address specified by its operand.

Example:

```c
int num = 25;
int *iptr;  //Pointer Operator (Indirection Operator *)
iptr = &num;
```

Example: A program executes the two operations.

```c
#include<iostream.h>
#include<conio.h>
void main( )
{
    int num;
    int *iptr;
    int val;

    num = 300;
iptr = &num;
    val = *iptr;

    cout<<" Value of num is :"<<num<<endl;
    cout<<" Value of pointer :"<<iptr<<endl;
    cout<<" Value of val :"<<val<<endl;
}
```

Output:

```
Value of num is : 300
Value of pointer : 0xbff64494
Value of val : 300
```

(Pointer Arithmetic:

- We can perform arithmetic operations on a pointer just as you can a numeric value.
- There are four arithmetic operators that can be used on pointers:
  - Increment ++
  - Decrement --
  - Addition +
Subtraction

- Example:

```c
int num, *iptr;
num = 9;
iptr = &num;
iptr++;cout<<iptr;
```

- The following operation can be performed on pointers.
  - We can add integer value to a pointer.
  - We can subtract an integer value from a pointer.
  - We can compare two pointers, if they point the elements of the same array.
  - We can subtract one pointer from another pointer if both point to the same array.
  - We can assign one pointer to another pointer provided both are of same type.

- The following operations cannot be performed on pointers.
  - Addition of two pointers.
  - Subtraction of one pointer from another pointer when they do not point to the same array.
  - Multiplication of two pointers.
  - Division of two pointers.

- A program to illustrate the pointer expression and pointer arithmetic.

```c
#include<iostream.h>
#include<conio.h>
void main()
{
int a, b, x, y;
int *ptr1, *ptr2;
a = 30;
b = 6;
ptr1 = &a
ptr2 = &b;
x = *ptr1 + *ptr2 - 6;
y = 6 - *ptr1 / *ptr2 + 30;
cout<<"Address of a = " <<ptr1<<endl;
cout<<"Address of b = " <<ptr2<<endl;
cout<<"a = " <<a<<""b = " <<b<<endl;
cout<<"x = " <<x<<""y = " <<y<<endl;

*ptr1 = *ptr1 + 70;
*ptr2 = *ptr2 * 2;
OUTPUT:
Address of a = 65524
Address of b = 65522
a = 30    b = 6
x = 30    y = 6
a = 100    b = 12
```

cout<<"a = ""<<a""""b = ""<<b""""endl;
}

**Pointers and Arrays:**

- There is a close relationship between array and pointers in C++.
- Consider the following program which prints the elements of an array A.

```cpp
#include<iostream.h>
void main()
{
    for (int i = 0; i<5; i++)
        cout<<A[i]<<"\t";
}
```
- Output of the above program is: 15 25 67 83 12
- When we declare an array, its name is treated as a constant pointer to the first element of the array.
- This is also known as the **base address of the array**.
- In other words base address is the address of the first element in the array of the address of a[0].
- If we use constant pointer to print array elements.

```cpp
#include<iostream.h>
void main()
{
    cout<< *(A) <<"\t";
    cout<< *(A+1) <<"\t";
    cout<< *(A+2) <<"\t";
    cout<< *(A+3) <<"\t";
    cout<< *(A+4) <<"\t";
}
```
- Output of the above program is: 15 25 67 83 12
- Here the expression *(A+3) has exactly same effect as A[3] in the program.
- The difference between constant pointer and the pointer variable is that the constant pointer cannot be incremented or changed while the pointer to an array which carries the address of the first element of the array may be incremented.
- The following example shows the relationship between pointer and one dimensional array.

```cpp
#include<iostream.h>
void main()
{
    cout<< "\n";
    cout<< *(A+3) <<"\t";
    cout<< *(A+2) <<"\t";
    cout<< *(A+1) <<"\t";
    cout<< *(A) <<"\t";
}
```
int a[10], i, n;
cout<<"Enter the input for array";
cin>>n;
cout<<"Enter array elements:";
for(i=0; i<n; i++)
    cin>>*(a+i);
cout<<"The given array elements are ":
for(i=0; i<n; i++)
cout<<"\t"<<*(a+i);
getch( );

➤ Array of Pointers:
   • There is an array of integers; array of float, similarly there can be an array of pointers.
   • “An array of pointer means that it is a collection of address”.
   • The general form of array of pointers declaration is:

   ```cpp
   int *pint[5];
   ```

   • The above statement declares an array of 5 pointers where each of the pointer to integer variable.
   • Example: Program to illustrate the array of pointers of isolated variables.

   ```cpp
   #include<iostream.h>
   #include<conio.h>
   void main( )
   {
       int *pint[5];
       int a = 10, b = 20, c = 30, d=40, e =50;
       pint[0] = &a;
       pint[1] = &b;
       pint[2] = &c;
       pint[3] = &d;
       pint[4] = &e;
       for( int  i=0; i<=4; i++)
           cout<<"Value “ << *pint[i]<< "stored at “<<pint[i]<<endl;
   }

   ➤ Pointers and strings:
   • String is sequence of characters ends with null (‘\0’) character.
   • C++ provides two methods of declaring and initializing a string.
   • Method 1:

   ```cpp
   char str1[ ] = “HELLO”;
   ```

   • When a string is declared using an array, the compiler reserves one element longer than the
number of characters in the string to accommodate NULL character.

- The string str1[ ] is 6 bytes long to initialize its first 5 characters HELLO\0.

- **Method 2:**
  
  ```
  char *str2 = “HELLO”;
  ```

- C++ treats string constants like other array and interrupts a string constant as a pointer to the first character of the string.

- This means that we can assign a string constant to a pointer that point to a char.

- **Example:** A program to illustrate the difference between strings as arrays and pointers.

  ```
  #include<iostream.h>
  #include<conio.h>
  void main( )
  {
      char str1[ ] = “HELLO”;  
      char *str2 = “HELLO”; 
      cout<<str1<<endl;      
      cout<<str2<<endl;      
      str2++;   
      cout<<str2<<endl;
  }
  ```

  **OUTPUT:**
  
  HELLO
  HELLO
  ELLO

- **Pointers as Function Parameters:**

- A pointer can be a parameter. It works like a reference parameter to allow change to argument from within the function.

```
void swap(int *m, int *n) 
{
    int temp;
    temp = *m;
    *m = *n;
    *n = temp;
} 
void swap(&num1, &num2); 
``` 

- **Pointers and Functions:**

- A function may be invoked in one of two ways:
  - Call by value
  - Call by reference

- The second method call by reference can be used in two ways:
  - By passing the references
  - By passing the pointers

- Reference is an alias name for a variable.

**Important 3 Marks**
• For Example:
  int m = 23;
  int &n = m;
  int *p;
  p = &m;

• Then the value of m i.e. 23 is printed in the following ways:
  cout <<m;    // using variable name
  cout << n;   // using reference name
  cout << *p;  // using the pointer

✓ Invoking Function by Passing the References:
• When parameters are passed to the functions by reference, then the formal parameters become references (or aliases) to the actual parameters to the calling function.
• That means the called function does not create its own copy of original values, rather, it refers to the original values by different names i.e. their references.
• For example the program of swapping two variables with reference method:

```
#include<iostream.h>
void main()
{
    void swap(int &, int &);
    int a = 5, b = 6;
    cout << "n Value of a:" << a << " and b:" << b;
    swap(a, b);
    cout << "n After swapping value of a:" << a << " and b:" << b;
}
void swap(int &m, int &n)
{
    int temp;
    temp = m;
    m = n;
    n = temp;
}
```

**OUTPUT:**

```
Value of a: 5 and b: 6
After swapping value of a: 6 and b: 5
```

✓ Invoking Function by Passing the Pointers:
• When the pointers are passed to the function, the addresses of actual arguments in the calling function are copied into formal arguments of the called function.
• That means using the formal arguments (the addresses of original values) in the called function; we can make changing the actual arguments of the calling function.
• For example the program of swapping two variables with Pointers:

```
#include<iostream.h>
```
void main()
{
    void swap(int *m, int *n);
    int a = 5, b = 6;
    cout << "Value of a :" << a << " and b :" << b;
    swap(&a, &b);
    cout << "After swapping value of a :" << a << " and b :" << b;
}
void swap(int *m, int *n)
{
    int temp;
    temp = *m;
    *m = *n;
    *n = temp;
}

Memory Allocation of pointers:
- Memory Allocation is done in two ways:
  - Static Allocation of memory
  - Dynamic allocation of memory.

Static Allocation of Memory:
- Static memory allocation refers to the process of allocating memory during the compilation of the program i.e. before the program is executed.
- Example:
  ```
  int a; // Allocates 2 bytes of memory space during the compilation.
  ```

Dynamic Allocation of Memory:
- Dynamic memory allocation refers to the process of allocating memory during the execution of the program or at run time.
- Memory space allocated with this method is not fixed.
- C++ supports dynamic allocation and de-allocation of objects using new and delete operators.
- These operators allocate memory for objects from a pool called the free store.
- The new operator calls the special function operator new and delete operators call the special function operator delete.

new operator:
- We can allocate storage for a variable while program is running by using new operator.
- It is used to allocate memory without having to define variables and then make pointers point to them.

OUTPUT:
Value of a : 5 and b : 6
After swapping value of a : 6 and b : 5

Important 3 Marks
The following code demonstrates how to allocate memory for different variables.

To allocate memory type integer
```c
int *pnumber;
pnumber = new int;
```

The first line declares the pointer, pnumber. The second line then allocates memory for an integer and then makes pnumber point to this new memory.

To allocate memory for array, double *dptr = new double[25];

To allocates dynamic structure variables or objects, student sp = new student;

- **delete Operator:**
  - The delete operator is used to destroy the variables space which has been created by using the new operator dynamically.
  - Use delete operator to free dynamic memory as : delete iptr;
  - To free dynamic array memory: delete [] dptr;
  - To free dynamic structure, delete structure;

- **Difference between Static Memory Allocation and Dynamic Memory Allocation:**

<table>
<thead>
<tr>
<th>Sl no</th>
<th>Static Memory Allocation</th>
<th>Dynamic Memory Allocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Memory space is allocated before the execution of program.</td>
<td>Memory space is allocated during the execution of program.</td>
</tr>
<tr>
<td>2</td>
<td>Memory space allocated is fixed</td>
<td>Memory space allocated is not fixed</td>
</tr>
<tr>
<td>3</td>
<td>More memory space is required</td>
<td>Less memory space is required.</td>
</tr>
<tr>
<td>4</td>
<td>Memory allocation from stack area</td>
<td>Memory space form heap area.</td>
</tr>
</tbody>
</table>

- **Free store (Heap memory):**
  - Free store is a pool of memory available to allocated and de-allocated storage for the objects during the execution of the memory.

- **Memory Leak:**
  - If the objects, that are allocated memory dynamically, are not deleted using delete, the memory block remains occupied even at the end of the program.
  - Such memory blocks are known as orphaned memory blocks.
  - These orphaned memory blocks when increases in number, bring adverse effect on the system. This situation is called memory leak.
**Pointers and Structures:**

- We can create pointers to structures variable.
  ```c
  struct student
  {
    int roll_no;
    float fee;
  };
  student s;
  student * sp = &s;
  (*sp).roll_no = 14;
  ```
- The above statement can be written using the operator as →
  ```c
  sp->roll_no = 14;
  ```

**Pointers and Objects:**

- The Pointers pointing to objects are referred to as object pointers.
- **Declaration of pointers to object**
  ```c
  class_name * object-pointer;
  ```
- Here class_name is the name of the class, object-pointer is the pointer to an object of this class type.
- Example: employee * eptr
  ```c
  #include<iostream.h>
  #include<conio.h>
  class employee
  {
    private:
    int empno;
    char name[20];
    float salary;
  public:
    void read()
    {
      cout<<“Enter the Employee Number, Name and Salaray”<<endl;
      cin>>empno>>name>>salary;
    }
    void display()
    {
  ```
cout<<"Employee Number:"<<empno;
cout<<"Employee Name:"<<name;
cout<<"Employee Salary:"<<salary;

};
void main( )
{
    employee e, * ep;
    ep = &e;
    clrscr( );
    ep -> read( );
    ep -> display( );
    getch( );
}
• Here, employee is an already defined class. When accessing members of the class an object pointer, the arrow operator (\textarrow) is used instead of dot (\textperiodcentered) operator.

\textbf{this pointers:}
• Every object in C++ has access to its own address through an important pointer called this pointer.
• The “this pointer” is an implicit parameter to all member functions.
• Therefore, inside a member function, this may be used to refer to the invoking object.

\textbf{Program 12:} Create a class containing the following data members Register\_No, Name and Fees. Also create a member function to read and display the data using the concept of pointers to objects.
#include<iostream.h>
#include<conio.h>
class Student
{
    private:
        long regno;
        char name[20];
        float fees;
    public:
        void readdata( );
        void display( );
};
void Student::readdata( )
{
    cout<<"Enter the Register Number:"<<endl;
    cin>>regno;
cout<<"Enter the Student Name:"<<endl;
cin>>name;
cout<<"Enter the Fees:"<<endl;
cin>>fees;
}
void Student::display( )
{
    cout<<"Register Number : "<<regno<<endl;
    cout<<"Student Name : "<<name<<endl;
    cout<<"Fees : "<<fees<<endl;
}
void main( )
{
    Student *S; // Create a pointer to point Student object
    clrscr( );
    S->readdata( ); // Access Student data member using a pointer
    S->display( ); // Display data using a pointer
    getch( );
}

OUTPUT 1:

Enter the Register Number: 243050
Enter the Student Name: Keerthi
Enter the Fees: 14050
Register Number : 243050
Student Name : Keerthi
Fees : 14050

OUTPUT 2:

Enter the Register Number: 12345
Enter the Student Name: Akash
Enter the Fees: 25000
Register Number : 12345
Student Name : Akash
Fees : 25000

### Important Questions

#### 1 Marks Question:

1. Define pointer. [June 2016]
2. Write the declaration syntax for a pointer. [March 2015]
3. How do we initialize a pointer? [March 2016]
4. Write any one advantage of pointers. [June 2015, March 2017]
5. What is the purpose of new operator in C++? [June 2017]
6. What is a pointer variable?
7. Which is the address operator?
8. What is pointer operator?
9. What is static memory & dynamic memory?
10. What is free store?

3 Marks Question:
1. What are the advantages of pointers? [June 2016]
2. What are the operations performed on pointers? [March 2015, June 2017]
3. What is array of pointer? Give example. [June 2015]
4. Explain the new and delete operator in pointers. [March 2016]
5. Define:
   a. Pointer.
   b. Static memory allocation.
   c. Dynamic memory allocation
6. What is the relationship between pointers and arrays?
7. Explain with example call by reference.
8. Distinguish between static and dynamic memory allocation.
9. What is the relationship between pointers and objects? Give an example
10. Explain the use of “this pointer”.

**************************