

Lecture # 1

(Data Structure)

Objective of the Course

- Covering well-known data structures such as dynamic arrays, linked lists, stacks, queues, trees and graphs etc.
- To Prepare students for (and is a prerequisite for) the more advanced material students will encounter in later courses.
- Implementing data structures and classical computer science problems in C++.

Teaching Procedure

- Lectures
- Discussions
- Assignments (Important)
- Quizzes
- Sessional Exams
- Final Exam

Material / Resources

- Text Book
 - Fundamentals of Data Structures in C++ by horowitz, sahni and mehta.
- www
- Any other good book on Data Structure

Grading

- Assignments/..... 15%
- Quizzes 05%
- Sessionals..... 30%
- Final Exam 50%

Keys for success in the course.....

- 100 % Attendance
- Solving Assignments **Youselfes**
- Attentive in class sessions
- Asking **questions** & **questions** unless you are clear about your problem.

Revision of Pointers

- Every byte in the computer's memory has an *address*. Addresses are numbers just as every house in a locality has got an address.
- Your program, when it is loaded into memory, occupies certain range of these addresses. This means that every variable and every function in your program starts with a particular address
- **NOTE**: Pointer can only contain the ADDRESS of certain memory location.

- Normally a variable directly contains a specific value. Pointer, on the other hand, contains an address of the variable that contains the specific value.
- Pointers, like other variables must be declared before they are used. For e.g.
`int *pttrr, temp;`
- Pointers should be initialized either when they are declared or in an assignment statement. A pointer may be initialized to **0**, **NULL** or an **address**. A pointer with the value **0** or **NULL** points to nothing.

- **NULL** is a symbolic constant defined in the header file **<iostream>**. Initializing a pointer to **NULL** is equivalent to initializing a pointer to **0**.
- Initializing a pointer is necessary to prevent it pointing to unknown or un-initialized area of memory.
- There are two pointer operators;
 - the **address of** operator &
and
 - **indirection operator** or **dereferencing operator**
i.e. *****.

- & is a unary operator that returns the **address of** its operand. For e.g.

```
int x = 5;
```

```
int *ptrr = NULL;
```

```
ptrr = &x;
```

- In above * shows ptrr is pointer variable whose type is int i.e. it can only contain the **address of** any **integer variables** in memory.

Similarly

```
cout<< *ptrr <<"\n";
```

```
cout<< x <<"\n";
```

- Will produce the same result 5. Here ***** means **value of the variable to which ptrr points to, which is x.**
- Note that the dereferenced pointer may also be used on the left side of an assignment statement as follows.

```
*ptrr = 9; or
```

```
cin>>*ptrr;
```

- Following program explains the concept of pointers

```
#include<iostream>
```

```
Using namespace std;
```

```
void main( )
{
    int x = 5;
    int *ptrr = NULL;
    ptrr = &x;
    cout<< *ptrr <<"\n";
    cout<< x <<"\n";
    *ptrr = 9;
    cout<< &x <<"\n";
    cout<< *&ptrr <<"\n"; // cancel effect of each other ( & and *)
    cout<< &*ptrr <<"\n";
    cout<< ptrr <<"\n";
    cout<< *ptrr <<"\n";
    cout<< x <<"\n";
}
```

Output of Program

```
5
5
0x0064FDF4
0x0064FDF4
0x0064FDF4
0x0064FDF4
9
9
```

new and delete operators

- Variables created during execution (run time) of a program by means of special operation is known as *Dynamic Data*. In C++ operation is new.
- The new operation has two forms

new DataType

new DataType [intExpression]

First form is used for creating a single variable of type DataType (e.g. int,...) at *run time*.
Second form creates an array whose elements are of type DataType (e.g. char,...) at *run time*.

■ Example

```
int *intptr;  
char *namestr;  
intptr = new int;  
namestr = new char[8];
```

- Variables created by **new** are said to be on the **free store** or **heap**, a region of memory set aside for **Dynamic Variables**.
- The **new** operator obtains a chunk (portion) of memory from the **free store**.

- A **Dynamic Variable** is unnamed and cannot be directly addressed. It must be indirectly addressed through the pointer returned by the new operator.

```
int *intptr = new int;  
char *namestr = new char[8];  
*intptr = 357;  
strcpy( namestr, "datastructure" );
```

- Dynamic data can be **destroyed** at any time during the **execution** of a program when it is no longer needed. The built-in operator **delete** does that and has two forms, **one** for deleting single variable, the **other** for deleting an array.

```
delete pointer;  
delete [ ] pointer;
```

■ Example :

```
int *ptr1 = new int;
```

```
int *ptr2 = new int;
```

```
*ptr2 = 44;
```

```
*ptr1 = *ptr2;
```

```
ptr1 = ptr2;
```

```
delete ptr2;
```

- **Inaccessible object :**

A dynamic variable on the free store without any pointer pointing to it.

- **Dangling pointer :**

A pointer that points to a variable that has been deallocated.

NOTE : *Leaving inaccessible objects on the free store should be considered a logic error.*

Implementation of new and delete operators

Example :

```
#include<iostream>

using namespace std;

struct node
{
    char info[15];
};

class trial
{
private :
    node obj1, *temp1, *temp2, *temp3;
    int I, length;
    char *p,*q;
public :
    trial();
    ~trial(); // Prototype
    void startin();
};
```

```
//-----
void main()
{
    clrscr();
    trial lnk;
    lnk.startin();

    getch();
}//-----
trial :: trial()
{
    temp1 = temp2 = temp3 = NULL;

}//-----
trial :: ~trial()
{
    delete temp1;
    delete temp2, temp3;
    delete[] p;          // delete 10 chars
}//-----
```

```
void trial :: startin()
{
    cout<<"\n Making use of \"new\" and \"delete\" is as follows.\n";
    cout<<"\n -----|n\n";
    temp1 = new node;
    temp2 = new node;
    cout<<"\n Enter information about temp1.\n";
    cin>>temp1->info;
    cout<<"\n Enter information about temp2.\n";
    cin>>temp2->info;

    temp3 = &obj1;
    cout<<"\n Enter information about temp3.\n";
    cin>>temp3->info;

    cout<<"\n Showing information of temp1.\n";
    cout<<temp1->info;
    cout<<"\n Showing information of temp2.\n";
    cout<<temp2->info;
    cout<<"\n Showing information of temp3.\n";
    cout<<temp3->info;
    cout<<"\n -----|n\n";
```

```
cout<<" Now enter the length of character array.\n";
cin>>length;

p = new char[length]; // allocate 10 chars
q = p;
cout<<" Now enter "<<length<<" characters to fill an array.\n";
for(int i=0;i<length; i++ )
{
    cin>>*p;
    p = p + 1;
}
p = q;
```

```
cout<<" \nElements of array are as follows.\n";
for(int i=0;i<length; i++ )
{
    cout<<*p<<", ";
    p = p + 1;
}
p = q;

}//--- END of startin( )-----
```

Thank You . . .