Practical Lecture : Pointer 2

Quick Recap

Let’s take a quick recap of previous lecture –

A)

B)

C)

D)

Today’s Agenda

Today we are going to cover -

- Wild pointer
- Null pointer
- Class and pointer
- This pointer

Let’s Get Started-

Wild pointer

Uninitialized pointers are known as wild pointers because they point to some arbitrary memory location and may cause a program to crash or behave badly.

```
int main()
{
int *p; /* wild pointer */

/* Some unknown memory location is being corrupted.

This should never be done. */

*p = 12;
}
```

Wild pointer

Please note that if a pointer p points to a known variable then it’s not a wild pointer. In the below program, p is a wild pointer till this points to a.

```
int main()
{
int *p; /* wild pointer */

int a = 10;

p = &a; /* p is not a wild pointer now*/

*p = 12; /* This is fine. Value of a is changed */
}
```

Null Pointer

NULL Pointer is a pointer which is pointing to nothing. In case, if we don't have address to be assigned to a pointer, then we can simply use NULL.

```
#include <iostream>
int main()
{
   // Null Pointer
   int *ptr = NULL;

   cout<<ptr
   return 0;
}
```

Null Pointer

Important Points

- NULL vs Uninitialized pointer – An uninitialized pointer stores an undefined value. A null pointer stores a defined value, but one that is defined by the environment to not be a valid address for any member or object.

- NULL vs Void Pointer – Null pointer is a value, while void pointer is a type

Pointers to Class Members in C++

```
class Simple
{
    public:
    int a;
};

int main()
{
    Simple obj;
    Simple* ptr;   // Pointer of class type
    ptr = &obj;

    cout << obj.a;
    cout << ptr->a;  // Accessing member with pointer
}
```

Pointers to Class Members in C++

Here you can see that we have declared a pointer of class type which points to class's object. We can access data members and member functions using pointer name with arrow -> symbol.

Pointer to Data Members of Class

```
datatype class_name::*pointer_name = &class_name::datamember_name ;
```

```
class Data
{
    public:
    int a;
    void print()
    {
        cout << "a is "<< a;
    }
};
```

Pointer to Data Members of Class

```
int main()
{
    Data d, *dp;
    dp = &d;     // pointer to object
    int Data::*ptr=&Data::a;   // pointer to data member 'a'
     d.*ptr=10;
    d.print();

    dp->*ptr=20;
    dp->print();
}
```

Output:-

a is 10

a is 20

Pointer to Members Function of Class

```
return_type (class_name::*ptr_name) (argument_type) = &class_name::function_name;

class Data
{
   public:
   int f(float)
   {
      return 1;
   }
};

int (Data::*fp1) (float) = &Data::f;   // Declaration and assignment
int (Data::*fp2) (float);       // Only Declaration
```

Pointer to Members Function of Class

```
int main()
{
   fp2 = &Data::f;   // Assignment inside main()
}
```

This pointer

The this pointer holds the address of current object, in simple words you can say that this pointer points to the current object of the class. Let's take an example to understand this concept.

```
class Demo {
private:
  int num;
  char ch;
public:
  void setMyValues(int num, char ch){
    this->num =num;
    this->ch=ch;
  }  void displayMyValues(){
    cout<<num<<endl;
    cout<<ch;
  }
```

This pointer

```
int main(){
  Demo obj;
  obj.setMyValues(100, 'A');
  obj.displayMyValues();
  return 0;
}
```

Here you can see that we have two data members num and ch. In member function setMyValues() we have two local variables having same name as data members name. In such case if you want to assign the local variable value to the data members then you won't be able to do until unless you use this pointer, because the compiler won't know that you are referring to object's data members unless you use this pointer.

QNA Time

Any Questions ??

Any Questions??

Thank You!

See you guys in next class.