Pointers

Today’s Agenda

Today we are going to cover -

- Pointers
- Difference b/w pointers and reference variables
- Void pointer
- Pointer to Pointer
- Wild pointer
- Null pointer
- Class and pointer
- This pointer

Let’s Get Started-

Pointers

Pointer is a variable in C++ that holds the address of another variable. They have data type just like variables, for example an integer type pointer can hold the address of an integer variable and an character type pointer can hold the address of char variable.

Syntax:-

```
data_type *pointer_name;
```

```
int *p, var
```

As I mentioned above, an integer type pointer can hold the address of another int variable. Here we have an integer variable var and pointer p holds the address of var. To assign the address of variable to pointer we use ampersand symbol (&).

```
p = &var;
```

Pointers

```
#include <iostream>
using namespace std;
int main(){
  //Pointer declaration
  int *p, var=101;
  //Assignment
  p = &var;

  cout<<"Address of var: "<<&var<<endl;
  cout<<"Address of var: "<<p<<endl;
  cout<<"Address of p: "<<&p<<endl;
  cout<<"Value of var: "<<*p;
  return 0;
}
```

Output

```
Address of var: 0x7fff5dfffc0c
Address of var: 0x7fff5dfffc0c
Address of p: 0x7fff5dfffc10
Value of var: 101
```

Void Pointers

A void pointer is a general-purpose pointer that can hold the address of any data type, but it is not associated with any data type.

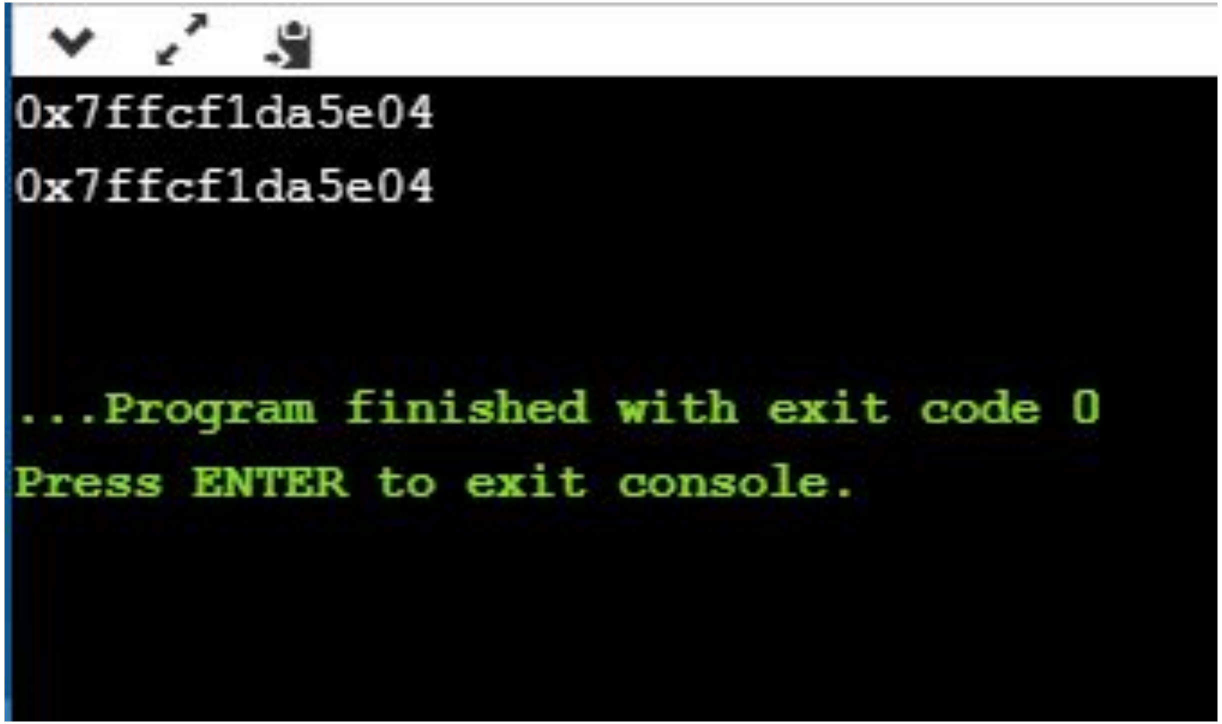
```
void *ptr;
```

Void Pointers

```
#include <iostream>
using namespace std;
int main()

{
  void *ptr;   // void pointer declaration
  int a=9;   // integer variable initialization
  ptr=&a;   // storing the address of 'a' variable in a void pointer variable.
  std::cout << &a << std::endl;
  std::cout << ptr << std::endl;
  return 0;
}
```

Output



```
0x7ffcf1da5e04
0x7ffcf1da5e04



...Program finished with exit code 0
Press ENTER to exit console.
```

Arithmetic Pointer

A pointer is an address which is a numeric value; therefore, you can perform arithmetic operations on a pointer just as you can a numeric value.

Incrementing a Pointer

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main () {
   int  var[MAX] = {10, 100, 200};
   int  *ptr;
   ptr = var;
   for (int i = 0; i < MAX; i++) {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;
      cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;
      ptr+; }
}
```

Output

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

Decrementing a Pointer

```
#include <iostream>
using namespace std;
const int MAX = 3;

int main () {
   int  var[MAX] = {10, 100, 200};
   int  *ptr;
   ptr = &var[MAX-1];
   for (int i = MAX; i > 0; i--) {
      cout << "Address of var[" << i << "] = ";
      cout << ptr << endl;
     cout << "Value of var[" << i << "] = ";
      cout << *ptr << endl;
      ptr--;
   }}
```

Decrementing a Pointer

```
Address of var[3] = 0xbfdb70f8
Value of var[3] = 200
Address of var[2] = 0xbfdb70f4
Value of var[2] = 100
Address of var[1] = 0xbfdb70f0
Value of var[1] = 10
```

Pointer to Pointer

We already know that a pointer points to a location in memory and thus used to store the address of variables. So, when we define a pointer to pointer. The first pointer is used to store the address of the variable. And the second pointer is used to store the address of the first pointer. That is why they are also known as double pointers.

Declaration:-

```
int **ptr;
```

Pointer to Pointer

```
#include<iostream>
using namespace std;
int main()
{
   int var = 789;
   int *ptr2;
   int **ptr1;
   ptr2 = &var;
   ptr1 = &ptr2;
   cout<< var<<endl;
   cout<<*ptr2<<endl ;
   cout<<**ptr1;

}
```

Output

```
789
789
789
```

MCQ 2

2. The operator used for dereferencing or indirection is ____

a) *

b) &

c) ->

d) –>>

MCQ 2

2. The operator used for dereferencing or indirection is ____

a) *

b) &

c) ->

d) –>>

Explanation: * is used as dereferencing operator, used to read value stored at the pointed address

MCQ 3

3.What will happen in the following C++ code snippet?

```
  int a = 100, b = 200;
  int *p = &a, *q = &b;
  p = q;
```

a) b is assigned to a

b) p now points to b

c) a is assigned to b

d) q now points to a

MCQ 3

3.What will happen in the following C++ code snippet?

```
  int a = 100, b = 200;
  int *p = &a, *q = &b;
  p = q;
```

a) b is assigned to a

b) p now points to b

c) a is assigned to b

d) q now points to a

Assigning to reference changes the object to which the reference is bound.

MCQ 4

4.Void pointer can point to which type of objects?

A)int

B) float

C) double

D) all of the mentioned

MCQ 4

Void pointer can point to which type of objects?

A)int

B) float

C) double

D) all of the mentioned

Predict the ouput 1

```
#include <iostream>
   using namespace std;

   int main()
   {
       char arr[20];
       int i;
       for(i = 0; i < 10; i++)
           *(arr + i) = 65 + i;
       *(arr + i) = '\0';
       cout << arr;
       return(0);
   }
```

Ouput

```
ABCDEFGHIJ
```

Each time we are assigning 65 + i. In first iteration i = 0 and 65 is assigned. So it will print from A to J.

Predict the ouput 2

```
#include <iostream>
  using namespace std;

  int main()
  {
      int a = 5, b = 10, c = 15;
      int *arr[ ] = {&a, &b, &c};
      cout << arr[1];
      return 0;
  }
```

Ouput

it will return some random number

Array element cannot be address of auto variable. It can be address of static or extern variables.

Predict the ouput 3

```
#include <iostream>
using namespace std;

int main()
{
   int a = 32, *ptr = &a;
   char ch = 'A', &cho = ch;

   cho += a;
   *ptr += ch;
   cout << a << ", " << ch << endl;
   return 0;
}
```

Ouput

```
129, a
```

Predict the ouput 4

```
#include <iostream>
using namespace std;

int main()
{
    int arr[] = { 4, 5, 6, 7 };
    int* p = (arr + 1);
    cout << *arr + 10;
    return 0;
}
```

Ouput

```
14
```

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 = nullptr;

   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.

Coding Question

1. Write a program to print all the alphabets using a pointer.

2. Write a program to print the elements of an array in reverse order using pointer.

3. Write a program to count the number of vowels and consonants in a string using a pointer.

4. Write a program to find the maximum number between three numbers using a pointer.

QNA Time

Any Questions??

Thank You!

See you guys in next class.