Pointers

Instructor : Isra Naz

CLO Covered

- CLO1: Describe fundamental problem-solving techniques and logic constructs. GA 1
- CLO2: Apply basic programming concepts. GA2

Memory and References

- Computer memory location is a collection of different consecutive memory locations.
- Each variable you declared in your program is assigned a unique location in the computer's memory known as address.
- A program may declare many variables for different tasks.
  - The variable name is used to refer to that memory location
  - It allows the user to access a value in the memory
- The computer refers to the memory using an address.

Memory and References

- A variable declaration associates three attributes to a variable
  - Variable name
  - Variable type
  - Variable memory address
- The following statement declares an integer variable;

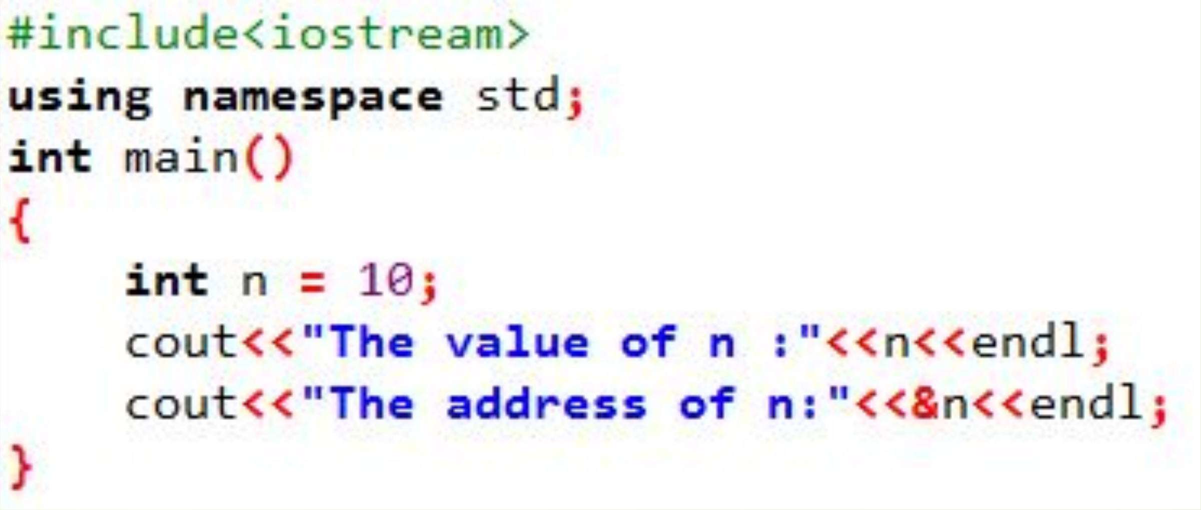
  int a;
- In this statement
  - Variable name is a
  - Type of the variable is int
  - But the address of the variable is unknown.

Address Operator &

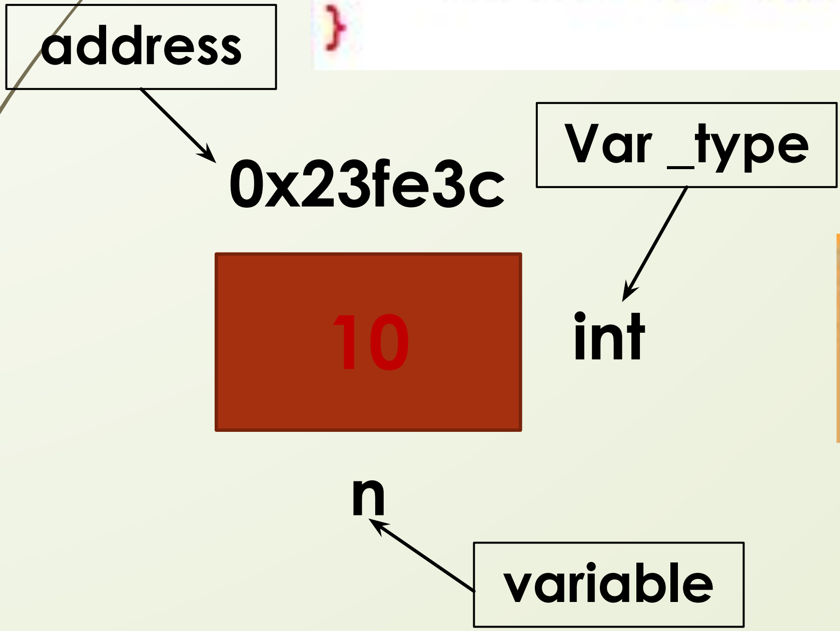
- Computer creates the variable at any available location in the memory.
- The address is a numerical number (often expressed in hexadecimal) that refers to particular location in the memory.
- To know where the data is stored, reference operator & is used.
- It is also known as address operator.
- The following statement will display the address of a

```
cout<<&a;
```

Example



```
#include<iostream>
using namespace std;
int main()
{
    int n = 10;
    cout<<"The value of n :"<<n<<endl;
    cout<<"The address of n:"<<&n<<endl;
}
```



address

0x23fe3c

Var _type

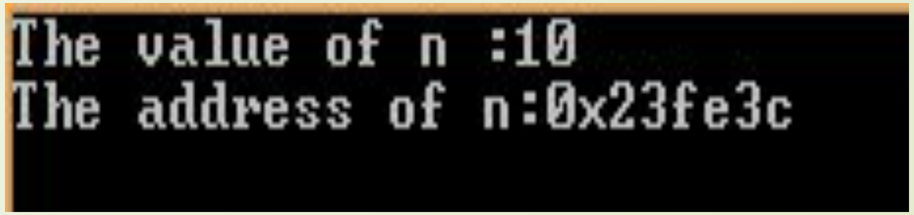
10

int

n

variable

Output



```
The value of n :10
The address of n:0x23fe3c
```

Pointers

- A variable that is used to store the memory address is called pointer variable or simply pointer
- Usually, a pointer is used to store the memory address of another variable that contain the actual value.
- The data type of pointer and variable whose address pointer is to be stored must match.

Pointers Declaration

- Pointer variable are declared in similar way as ordinary variables, except an asterisk (*).
- **Syntax**

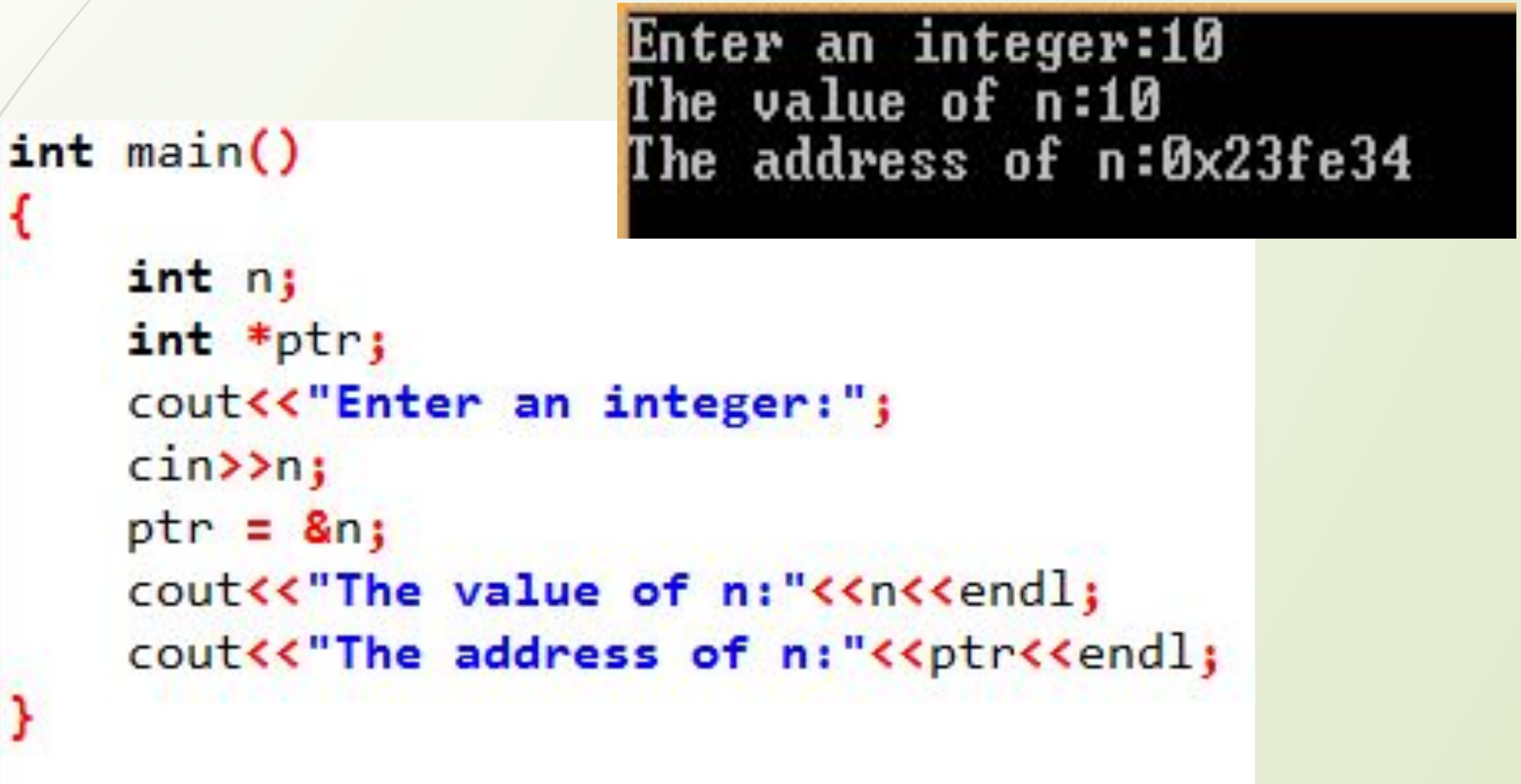
```
        dataType *var;
```

  - The * (asterisk) indicate that the variable is a pointer variable
- **Example**

```
int *p;
float *p1, *p2;
double *ptr1, *ptr2;
void *p;
```

Example

Output



```
int main()
{
    int n;
    int *ptr;
    cout<<"Enter an integer:";
    cin>>n;
    ptr = &n;
    cout<<"The value of n:"<<n<<endl;
    cout<<"The address of n:"<<ptr<<endl;
}
```

```
Enter an integer:10
The value of n:10
The address of n:0x23fe34
```

Pointer Initialization

- The pointers can also be initialized at the time of its declaration.
- C++ does not initialize variables automatically
- Therefore, a pointer variable should be initialized so that it may not point to anything invalid.
- A pointer initialized to 0, NULL or memory address of another variable.
- The value of 0 and NULL are equivalent.

Pointer Initialization

- **Syntax**

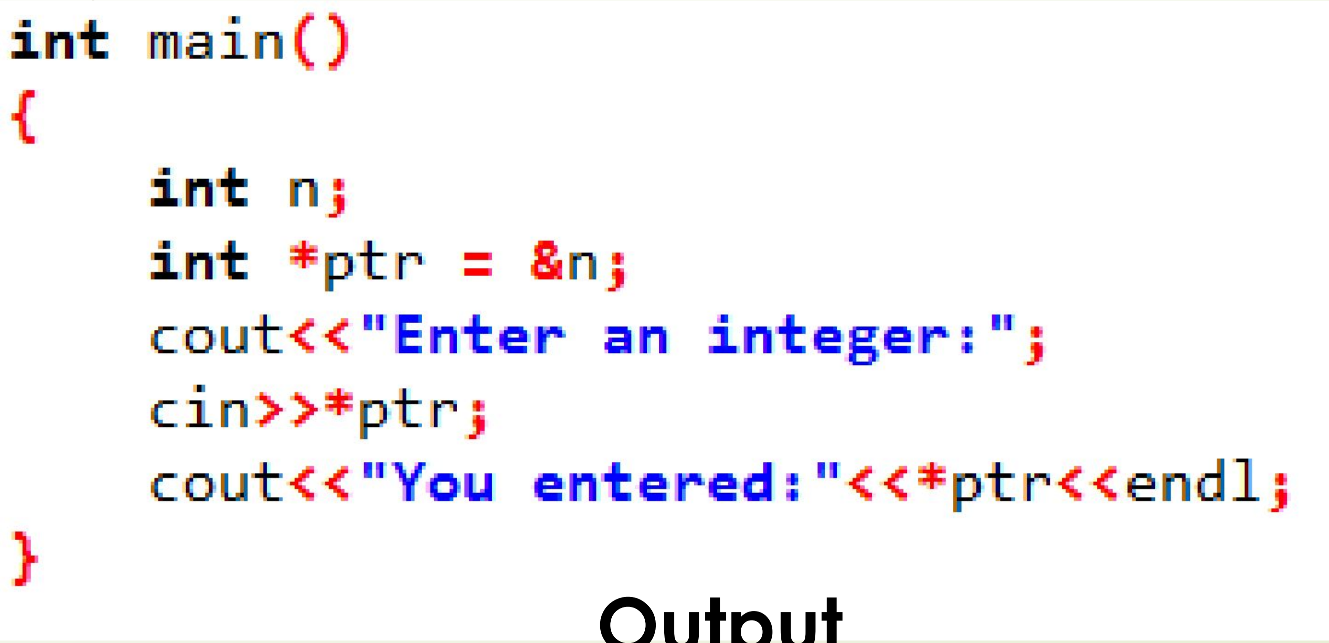
  ```
  DataType *PointerVariable = &RefVariable;
  ```

- **Example**

  ```
  int n = 100;
  int *p1 = &n;
  int *p2 = Null;
  ```

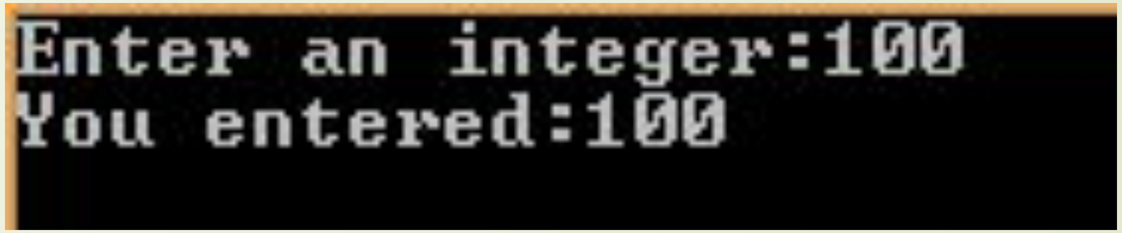
- In this example pointer p1 is initialized to the memory address of variable n.
- The pointer p2 is initialized to NULL.

Example



```
int main()
{
    int n;
    int *ptr = &n;
    cout<<"Enter an integer:";
    cin>>*ptr;
    cout<<"You entered:"<<*ptr<<endl;
}
```

Output



```
Enter an integer:100
You entered:100
```

NULL pointer

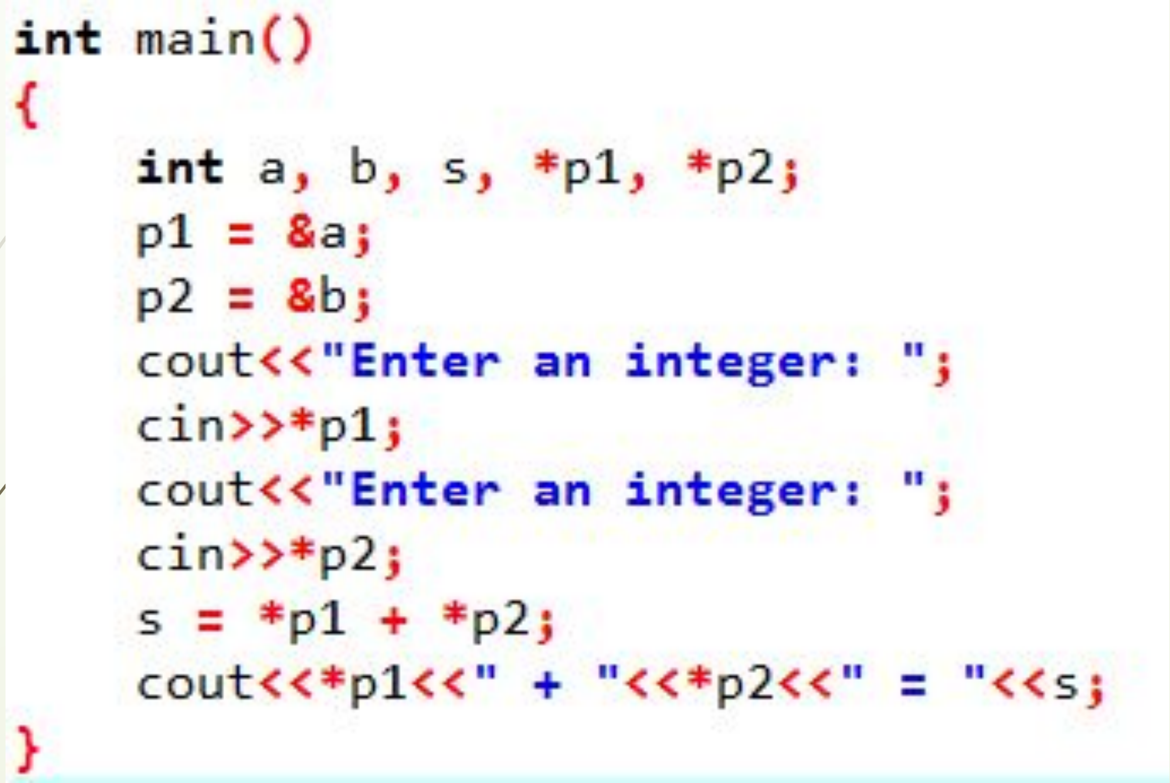
- NULL is a special value that indicates an empty pointer
- If you try to access a NULL pointer, you will get an error

```
int *p;
p = 0;
cout << p << endl; //prints 0
cout << &p << endl;//prints address of p
cout << *p << endl;//Error!
```

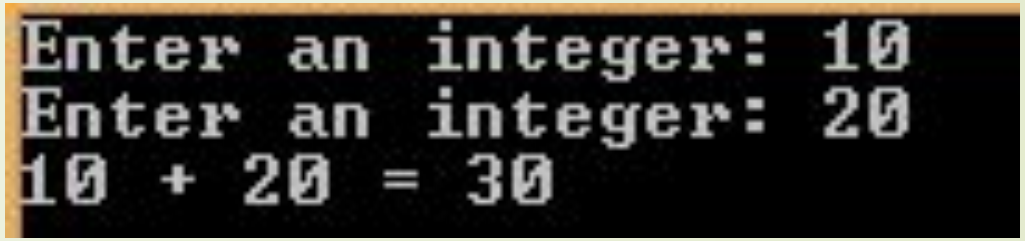
Dereference Operator

- It is used to access the value of the variable whose memory address is stored in pointer.
- It is denoted by asterisk * .
- It is also called indirection operator.
- It can also be used to input value in the variable and process the data stored in the variable.

Example



```
int main()
{
    int a, b, s, *p1, *p2;
    p1 = &a;
    p2 = &b;
    cout<<"Enter an integer: ";
    cin>>*p1;
    cout<<"Enter an integer: ";
    cin>>*p2;
    s = *p1 + *p2;
    cout<<*p1<<" + "<<*p2<<" = "<<s;
}
```



```
Enter an integer: 10
Enter an integer: 20
10 + 20 = 30
```

Pointer Arithmetic

- The arithmetical operations on pointers work differently than normal integer data types.
- Only addition and subtraction operations can be performed on pointers.
- The effect of both addition and subtraction depends on the size of the data type of the pointer.

Pointer Arithmetic

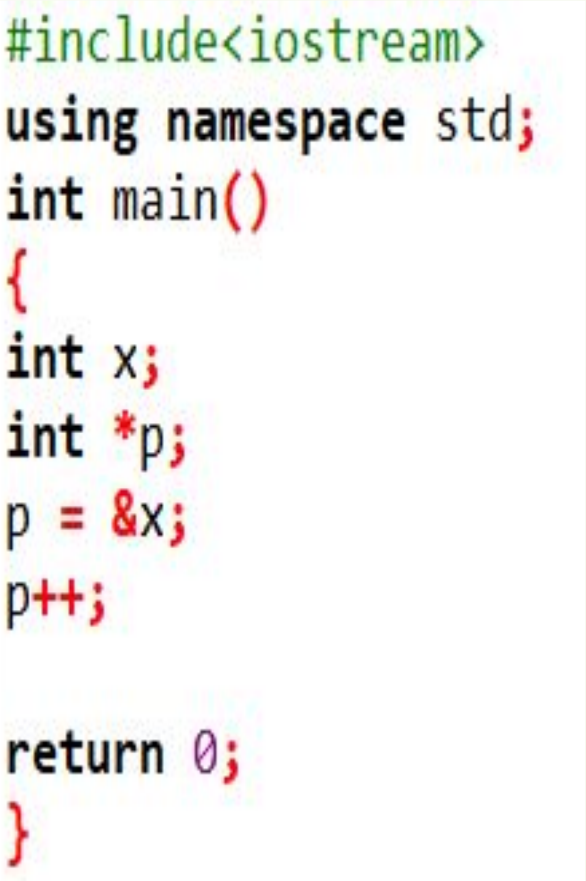
- When we add 1 to a pointer, we are actually adding the size of data type in bytes, the pointer is pointing at.
- For Example.

```
int *x;

x++;
```

- If current address of x is 1000, then x++ statement will increase x by 4(size of int data type) and makes it 1004, not 1001.
- If the increment operator is used with a char pointer, it will change the reference by 1 bytes.

Pointer Arithmetic (cont'd)



```
#include<iostream>
using namespace std;
int main()
{
int x;
int *p;
p = &x;
p++;

return 0;
}
```

201+1*4 = 205

	X					
200	201	202	203	204	205	206

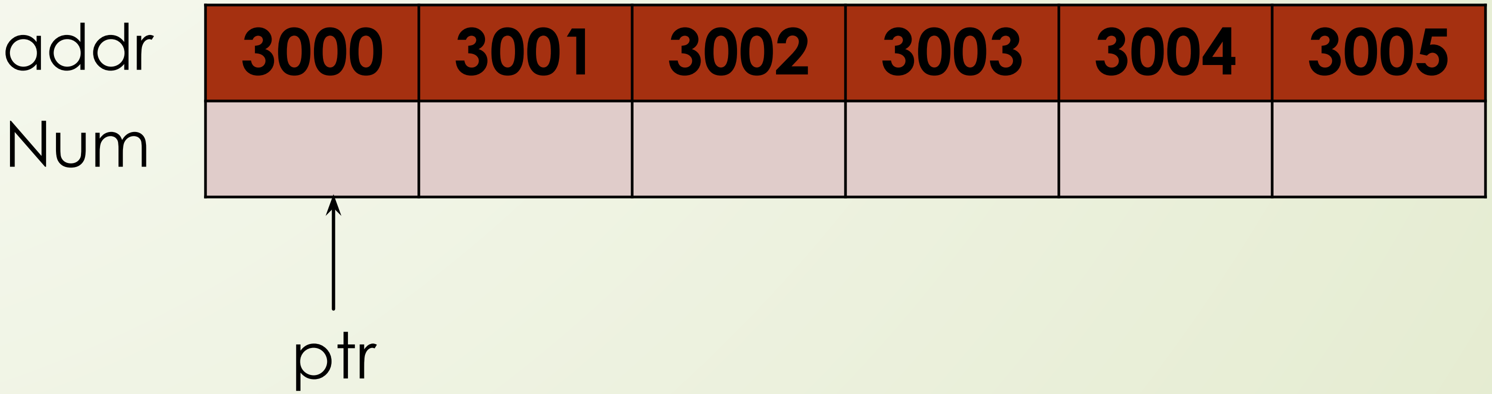
Pointers and Arrays

- The pointers can also be used with arrays.
- A pointer can access all elements of an array if the address of first element is assigned to it.
- The name of array represents the address of its first element.
- The address of first element can be assigned to a pointer by assigning the name of the array to pointer.
- The pointer then can access the remaining elements as they are stored consecutively in the memory.

Pointers and Arrays

- For Example, using pointer to access an array

```
int Num[10];
int *ptr;
ptr = Num;
```



addr	3000	3001	3002	3003	3004	3005
Num						

Accessing array elements with pointers

- The array elements can be accessed with pointers by moving the pointer to the desired element
- The contents of an array elements can be accessed using dereference operator * .
- The pointer reference can be move forward and backward by using increment operator ++ and decrement operator -- .

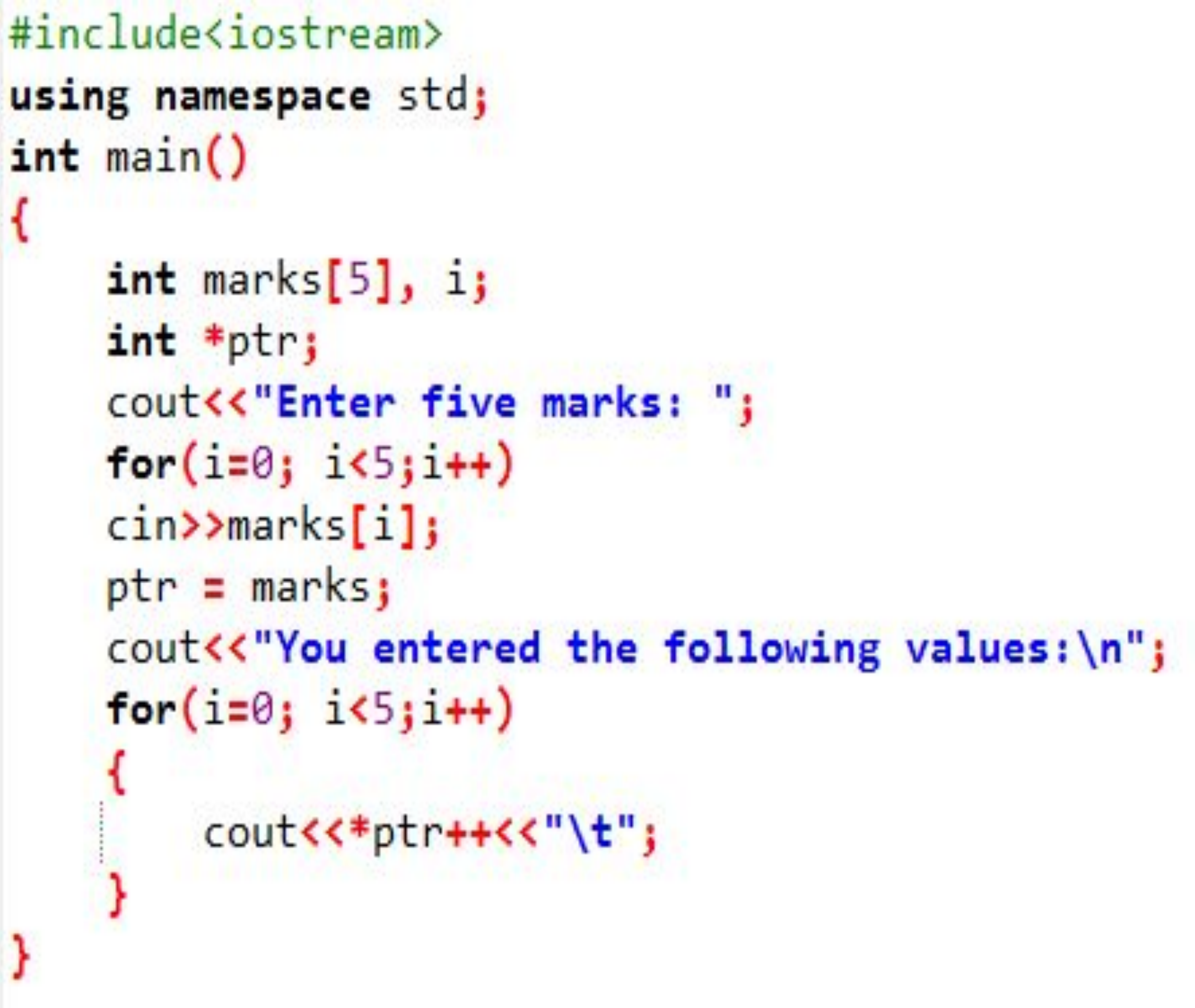
Accessing array elements with pointers

- **Example**

  -
    ```
    int Num[5] = {10, 20, 30, 40, 50};
    int *ptr = Num;
    cout<<*ptr;
    ptr++;
    cout<<*ptr;
    ```

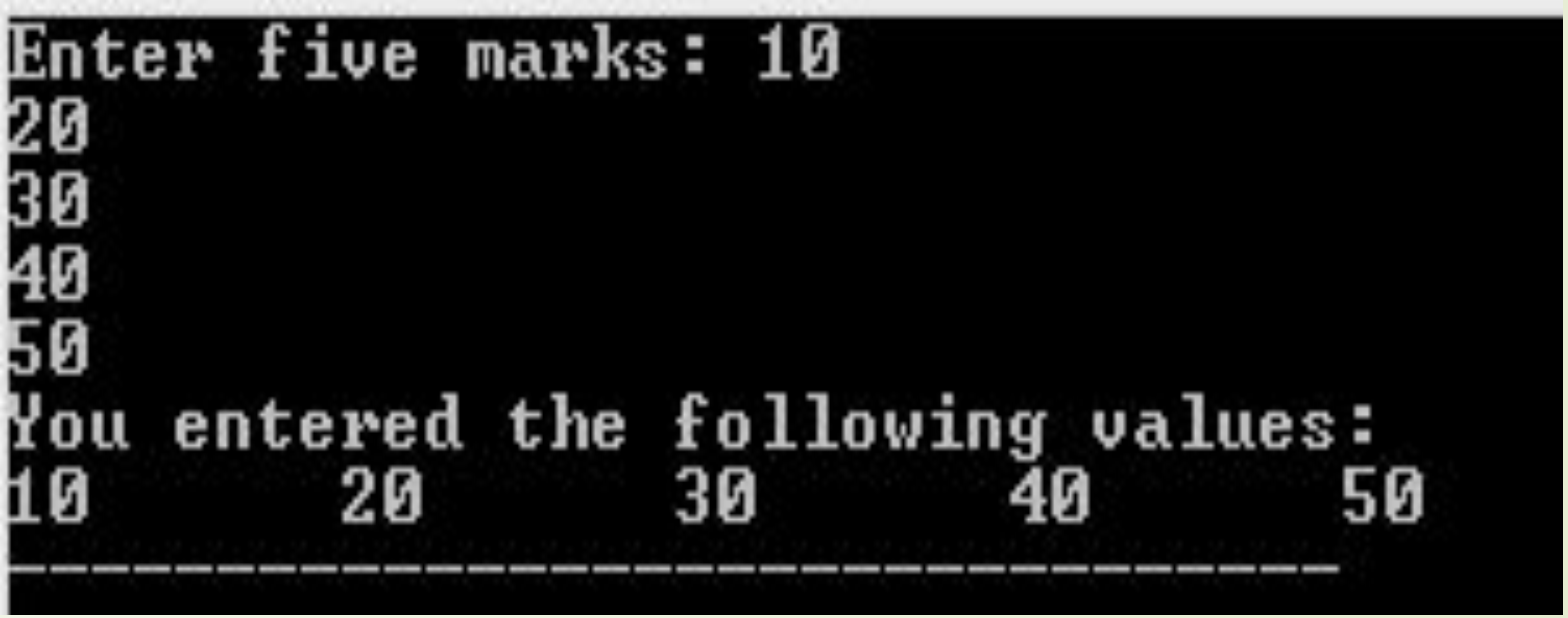
  -
    ```
    cout <<*ptr;
    cout<<*(ptr+1);
    cout<<*(ptr+2);
    ```

Example



```
#include<iostream>
using namespace std;
int main()
{
    int marks[5], i;
    int *ptr;
    cout<<"Enter five marks: ";
    for(i=0; i<5;i++)
    cin>>marks[i];
    ptr = marks;
    cout<<"You entered the following values:\n";
    for(i=0; i<5;i++)
    {
        cout<<*ptr++<<"\t";
    }
}
```

Output

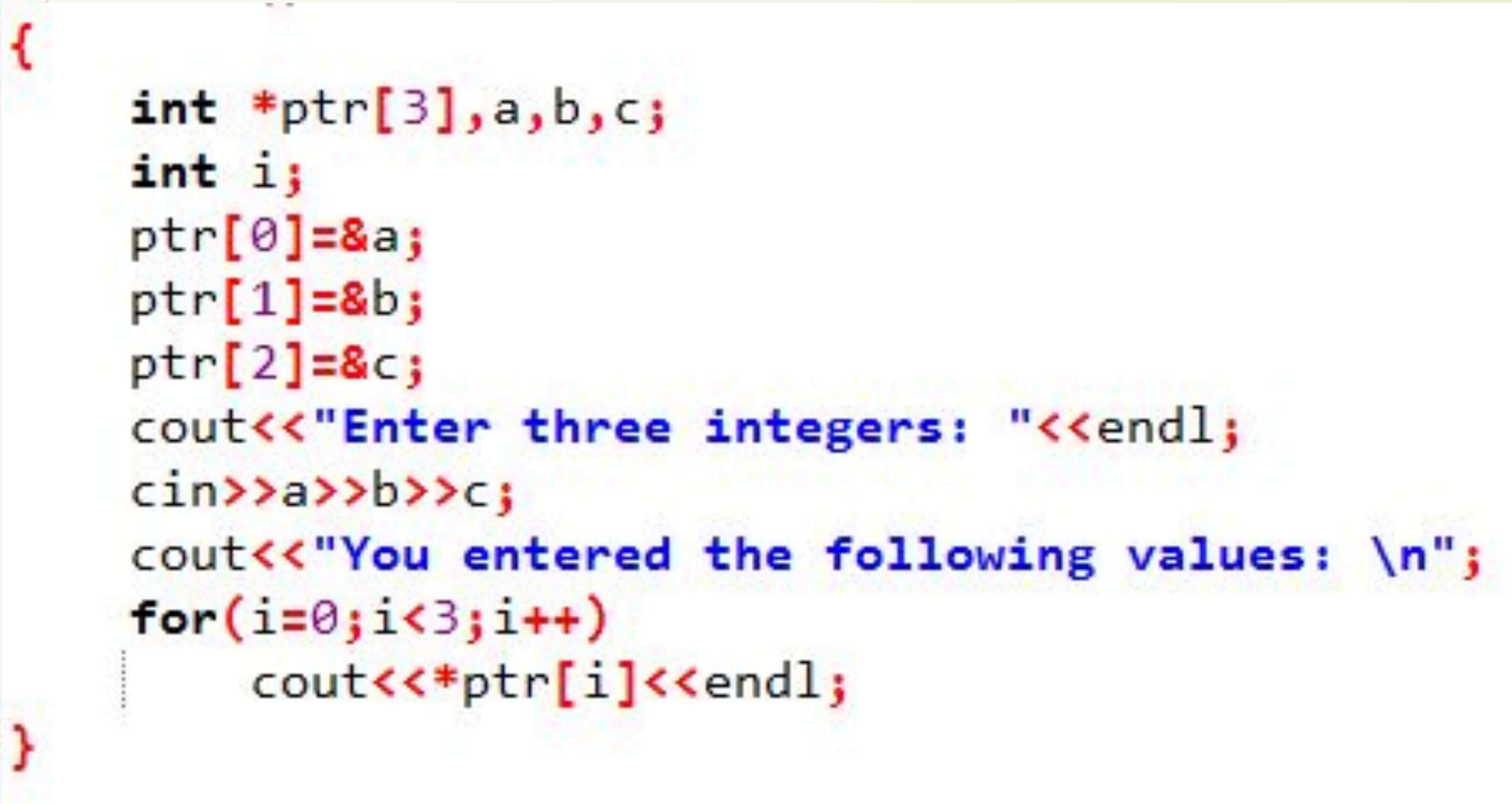


```
Enter five marks: 10
20
30
40
50
You entered the following values:
10      20      30      40      50
--------------------------------
```

Array of Pointers

- An array of pointers is an array in which each element is a pointer
- Each element in the array can store a memory address.
- The array can store the memory addresses of different objects of same type.

Example

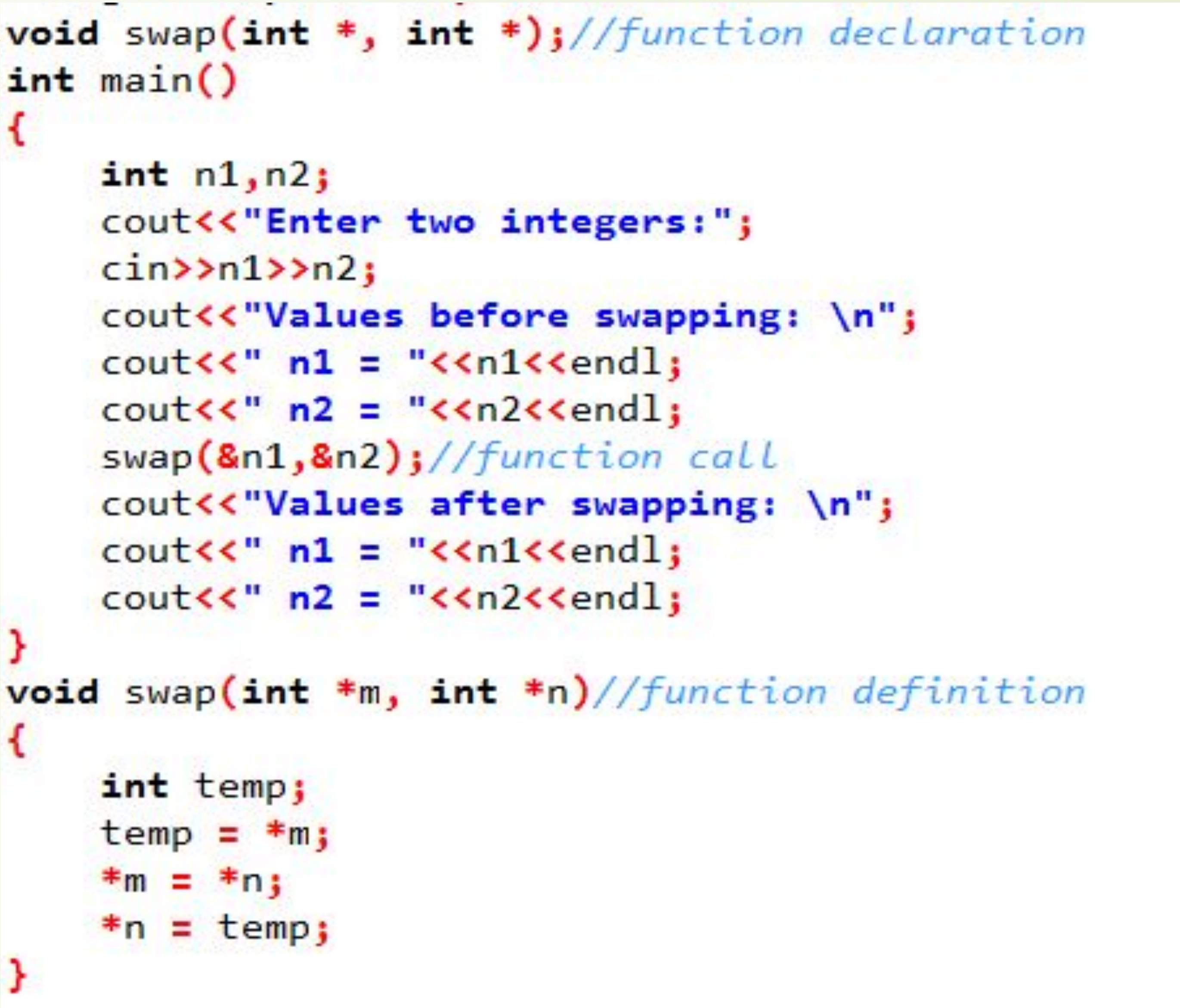


```
{
    int *ptr[3],a,b,c;
    int i;
    ptr[0]=&a;
    ptr[1]=&b;
    ptr[2]=&c;
    cout<<"Enter three integers: "<<endl;
    cin>>a>>b>>c;
    cout<<"You entered the following values: \n";
    for(i=0;i<3;i++)
        cout<<*ptr[i]<<endl;
}
```

Pointers and Functions

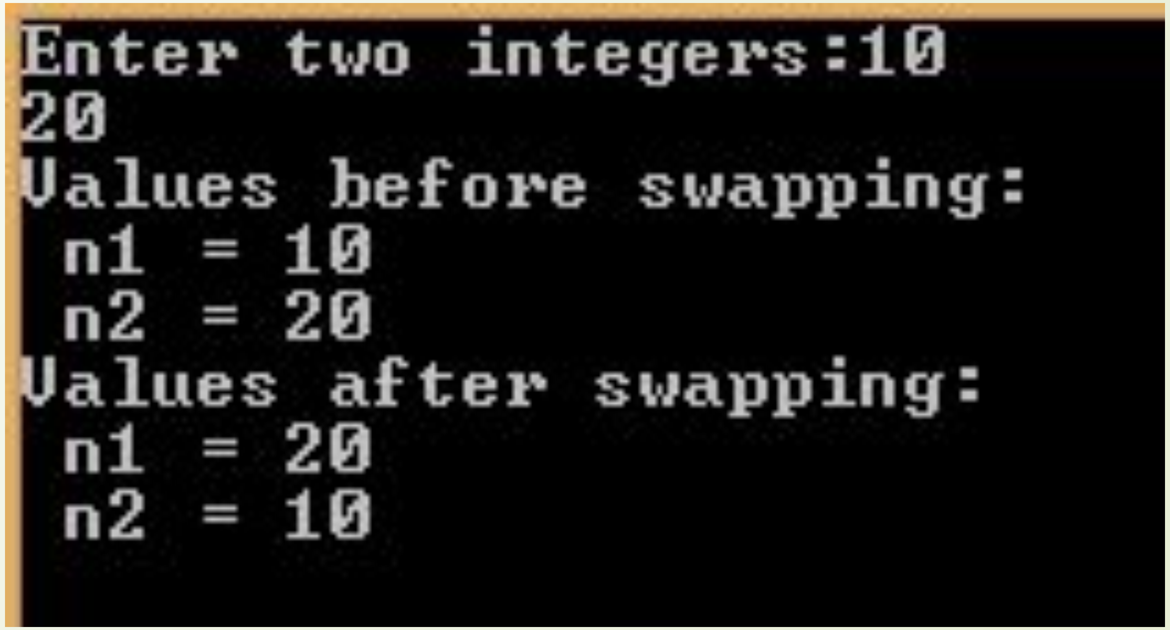
- The parameter can be passed to function using pointers.
- The address of actual parameter is passed to the formal parameter if the formal parameters are defined as pointers
- It is similar to passing parameters to a function by reference.
- Any change made in formal parameter by function actually changes the value of actual parameter in both cases.

Example



```
void swap(int *, int *);//function declaration
int main()
{
    int n1,n2;
    cout<<"Enter two integers:";
    cin>>n1>>n2;
    cout<<"Values before swapping: \n";
    cout<<" n1 = "<<n1<<endl;
    cout<<" n2 = "<<n2<<endl;
    swap(&n1,&n2);//function call
    cout<<"Values after swapping: \n";
    cout<<" n1 = "<<n1<<endl;
    cout<<" n2 = "<<n2<<endl;
}
void swap(int *m, int *n)//function definition
{
    int temp;
    temp = *m;
    *m = *n;
    *n = temp;
}
```

Output



```
Enter two integers:10
20
Values before swapping:
 n1 = 10
 n2 = 20
Values after swapping:
 n1 = 20
 n2 = 10
```

Memory Management with Pointers

- The process of allocating and de-allocating memory is known as memory management.
- **Static Memory Allocation**
  - Memory is allocated at compilation time
- **Dynamic Memory**
  - Memory is allocated at running time

Static vs. Dynamic Objects

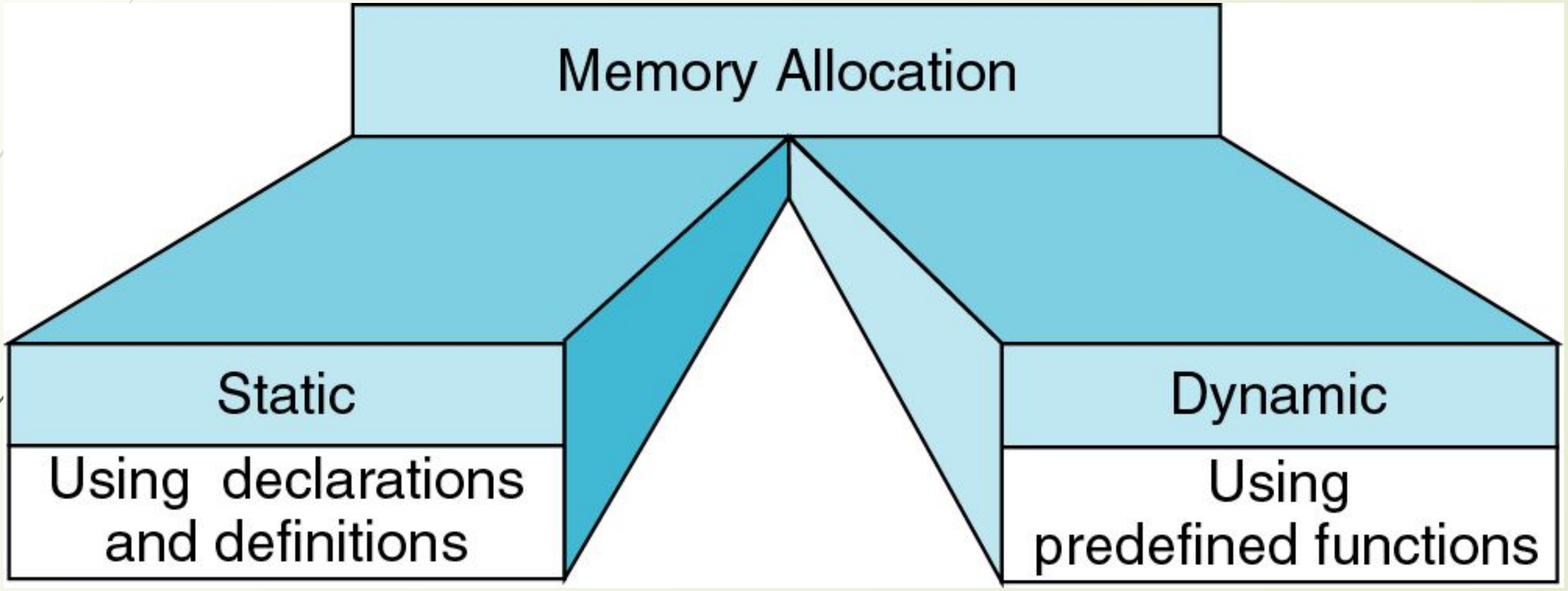
- **Static object**

  (variables as declared in function calls)

  - Memory is acquired automatically
  - Memory is returned automatically when object goes out of scope

- **Dynamic object**
  - Memory is acquired by program with an allocation request
    - `new` operation
  - Dynamic objects can exist beyond the function in which they were allocated
  - Object memory is returned by a deallocation request
    - `delete` operation

Memory Allocation


Memory Allocation
Static
Using declarations and definitions
Dynamic
Using predefined functions

new
delete

```
{
   int a[200];
   …
}
```

```
int* ptr;
ptr = new int[200];
…
delete [] ptr;
```

The New Operator

- The new operator is used to allocate memory dynamically.
- The compiler allocates the amount of memory according to the type of object.
- The new operator returns a memory address.
- The returned address must be assigned to a pointer.
- The pointer then access the memory location and process the values stored in that address.
- The new operator can be used to create simple variable, an object or an array of objects.

The New Operator

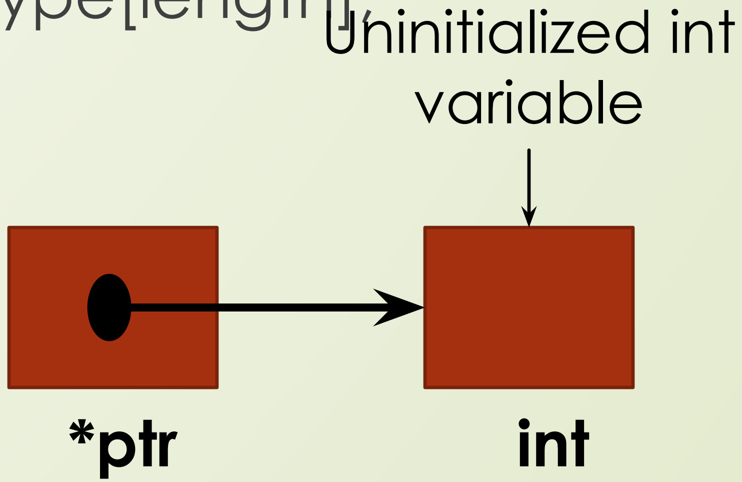
- **Syntax**
    - To create one variable

      new DataType;
    - To create an array dynamically

      New Datatype[length];
- **Example**

  int *ptr;

  ptr = new int;



The Delete Operator

- The delete operator de-allocates the memory and returns the allocated memory back to the free store.
- **Syntax**
  - To delete one variable dynamically

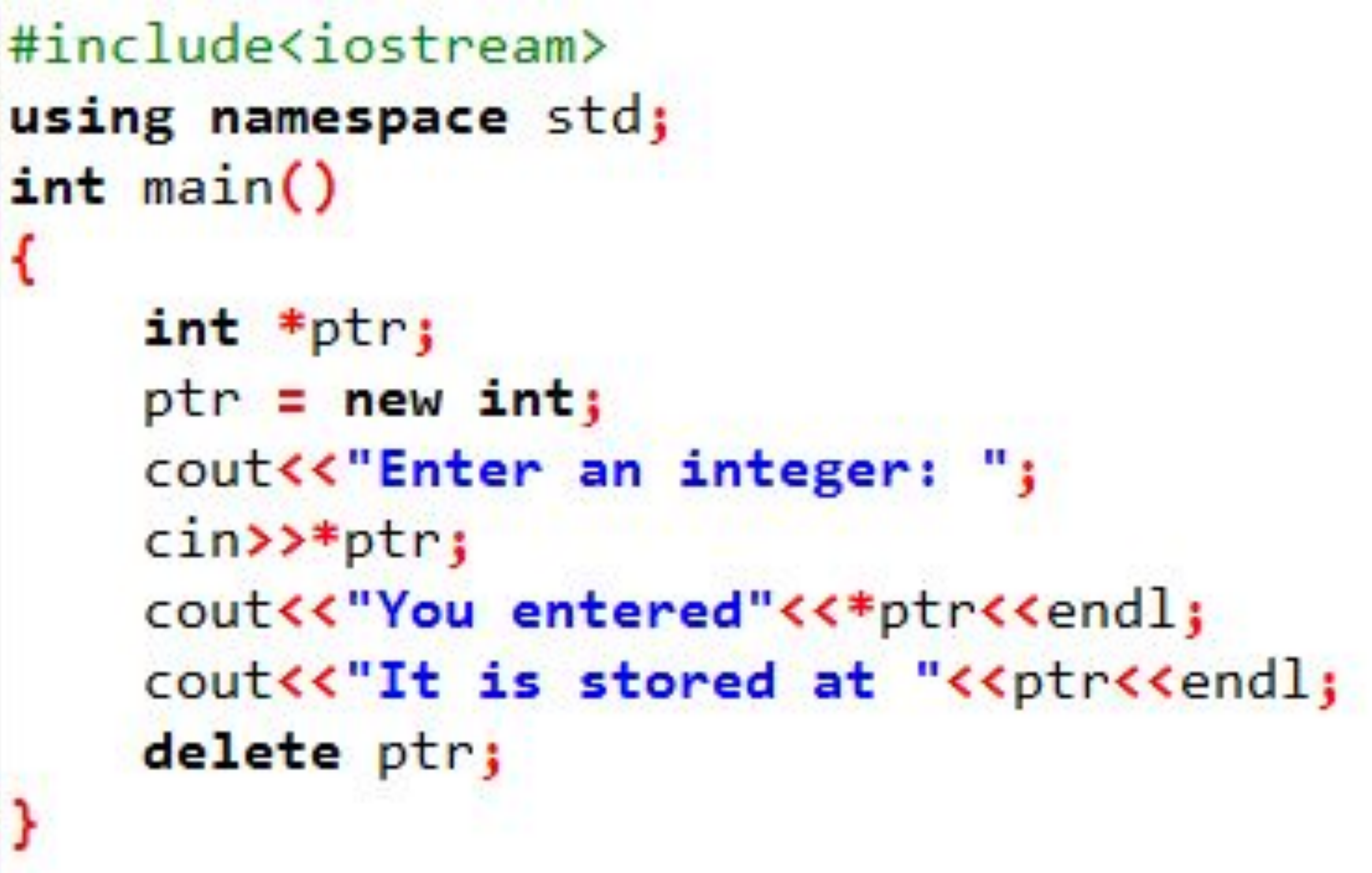
    delete variable;
  - To delete an array dynamically

    delete[ ] variable;
- **Example**

  delete ptr;

  It deallocates the memory referred by the pointer ptr

Example



```
#include<iostream>
using namespace std;
int main()
{
    int *ptr;
    ptr = new int;
    cout<<"Enter an integer: ";
    cin>>*ptr;
    cout<<"You entered"<<*ptr<<endl;
    cout<<"It is stored at "<<ptr<<endl;
    delete ptr;
}
```

Lecture End