Arrays, Pointer, References, and the Dynamic Allocation Operators

Array and Objects

See Program 12.1

Pointer to Objects

➔ Operator can be used to call member's of a class.

for e.g.:-

<class name> <object>, *pointer;

pointer = &object;

pointer->class_members;

See Program

"this" Pointer

"this" pointer is a pointer that is passed explicitly to a member function, this pointer points to the object that has invoked the member function.

A member function can access any data members of the class without the use of any object.

This is due to the "this" pointer, It points to the object that has invoked the function and exists only within the scope of the function.

See program 12.2.

Pointer to Derived Class

As we already know pointer of one data type cannot point to data of another type.

But There is an exception to this rule.

A Base class object's pointer can point to the derived class object. But the Base class cannot pointer cannot point to the data members that were not Inherited.

For e.g.:- See Program 12.3

It is possible to convert one data type to another by type casting by doing so we can even access non-inherited members of derived class

For e.g. :-

Arrays, Pointer, References, and the Dynamic Allocation Operators

Pointer = (int *)Pointer //converting Pointer to int data type

it is important to know that how objects look at functions and data members of a class.
1. Each object gets its own copy of the data member.
2. All access the same function definition as present in the code segment.
Meaning each object gets its own copy of data members and all objects share single copy of member functions.

Pointer to Class Members

C++ allows you to make pointer, that points to class members instead of pointing to instance of that members in a object.

To access that member, we have to use a special operators i.e. "**.***" and "**->***"

For E.g. – See Program 12.4

Pointers vs References in C++

What is Pointer?

Pointer is a variable that stores the address of another variable. We have to dereference the pointer with ***** to access the value at the address.

What is Reference ?

Reference is a variable that is just an alias, that is same variable with different name, You can think of Reference as a constant pointer(not to be confused with pointer to a constant value).You do not need to apply ***** operator to access value.

```
int i = 3;


// A pointer to variable i (or stores
// address of i)
int *ptr = &i;


// A reference (or alias) for i.
int &ref = i;
```

Arrays, Pointer, References, and the Dynamic Allocation Operators

See program 12.4.1

Differences :

1. **Initialisation:** A pointer can be initialised in this way:

```
 int a = 10;

  int *p = &a;

         OR

     int *p;

   p = &a;

we can declare and initalise pointer at same step or in multiple
line.
```

   While in refrences,

```
int a=10;

int &p=a;  //it is correct

   but

int &p;

 p=a;    // it is incorrect as we should declare and initialise
references at single step.
```

   NOTE: This differences may vary from compiler to compiler. Above differences is with respect to turbo IDE.

8. **Reassignment:** A pointer can be re-assigned. This property is useful for implementation of data structures like linked list, tree, etc. See the following examples:

```
int a = 5;

  int b = 6;

  int *p;

  p =  &a;

  p = &b;
```

   On the other hand, a reference cannot be re-assigned, and must be assigned at initialization.

```
int a = 5;

int b = 6;

int &p = a;

int &p = b;  //At this line it will show error as "multiple
declaration is not allowed".


However it is valid statement,

int &q=p;
```

14. **Memory Address:** A pointer has its own memory address and size on the stack whereas a reference shares the same memory address (with the original variable) but also takes up some space on the stack.
    NOTE However if we want true address of reference, then it can be found out in turbo IDE by writting the following statement,

```
    int &p = a;

    cout << &(&p);
```

17. **NULL value:** Pointer can be assigned NULL directly, whereas reference cannot. The constraints associated with references (no NULL, no reassignment) ensure that the underlying operations do not run into exception situation.
18. **Indirection:** You can have pointers to pointers offering extra levels of indirection. Whereas references only offer one level of indirection. I.e.,

```
In Pointers,

int a = 10;

int *p;

int **q;  //it is valid.

p = &a;

q = &p;


Whereas in refrences,


int &p = a;

int &&q = p; //it is reference to reference, so it is an error.
```

30. **Arithmetic operations:** Various arithmetic operations can be performed on pointers whereas there is no such thing called Reference Arithmetic.(but you can take the address of an object pointed by a reference and do pointer arithmetics on it as in &obj + 5).)

When to use What

The performances are exactly the same, as references are implemented internally as pointers. But still you can keep some points in your mind to decide when to use what :

- Use references
  - In function parameters and return types.
- Use pointers:
  - Use pointers if pointer arithmetic or passing NULL-pointer is needed. For example for arrays (Note that array access is implemented using pointer arithmetic).
  - To implement data structures like linked list, tree, etc and their algorithms because to point different cell,we have to use the concept of pointers.

  For Example,
  In Pointers

```
int arr[] = {10,20,30};
int *p,i ;
p = arr;
for(i = 0; i<3 ; i++)
{
cout << *p << endl;
p++;
}
```

Output:

```
10
20
30
```

NOTE: Here pointer will be ahead of base address

Now take same code in Reference

```
int arr[] = {10,20,30};
int &p = arr[0];
int i;
for(i = 0; i<3 ; i++)
{
cout << p << endl;
p++;  //here it will increament arr[0]++
}
```

Output:

```
10
11  // when p++ is run, the value comes out to be 11, which is
correct as we are actually adding 1 to 10
12
```

Passing Object References to the function

As we read in previous chapter, passing references of object instead object, will avoid creation of a copy object.

Returning References

Arrays, Pointer, References, and the Dynamic Allocation Operators

See Program 12.5

Limitation of References

- Cannot create Reference of a Reference
- Cannot obtain address of Reference
- Cannot create arrays of Reference
- Cannot create pointer of Reference
- Reference variable must be initialized when it is declared

Note :- A reference variable is known as independent reference.

E.g.:- int &ref = a;

Note :- In C++ int *a, b; = int* a, b; . Here b is not pointer in both cases.

C++ Dynamic Allocation Operators

In C++ **new** and **delete** operator are used to allocate and free memory. Just like in C **malloc()** and **free()** functions.

The new operator allocates and returns pointer to start of it.

Var = new type

delete Var

Var is a pointer variable that has enough memory to store address of a type type.

Advantages of new over free()

- Automatically allocates enough memory to hold an object of specified type, No need to use sizeof() as in C.
- Returns address of the memory, No need to use type conversion as in C.

Note :- Use new and delete operator together as they are compatible. Do not use new and free() as there is no guarantee they will work fine together.

Initializing allocated memory

Arrays, Pointer, References, and the Dynamic Allocation Operators

See program 12.6

Allocating Arrays

To allocate memory to object array.

See program 12.7

Note :- The class should have a default constructor because you cannot pass parameter to allocated object array.

Note :- If you want to throw NULL instead of a exception error if space is not allocated to the pointer, you have to use this :-

Int *p = new(nothrow) int;

Note :- The Placement Form of new and delete

Type *p = new(address to be returned(should already be allocated)) type;