C++ NOTE 6:

Pointers:

Pointers are a concept that are special to C++ and C. In languages such as java, the functioning and working of the pointers is done internally by the compiler. It takes away the hassle of pointers but also gives less control to the developer over the memory addresses of different components of the program. Let us understand what pointers are:

We know that we can get the memory address of any variable using the '&' symbol in-front of that variable. Now the address once retrieved might be used multiple times but we do not want to keep on adding & in front of variables all throughout our program, so we store that address into a variable which we can use to represent that particular address, that particular variable is pointer.

Definition: The pointer in C++ language is a variable, also known as locator or indicator that points to an address of a value.

Symbols to know while working with pointers:

i. & : Reference Operator/ Address of Operator
ii. * : Dereference Operator/ Star Operator/ Indirection Operator
iii. -> : Member Access Operator/ Arrow Operator

Declaration of a pointer: While declaring a pointer always remember that the type of the pointer should be same as the type of the variable whose address the pointer will store. For eg.

```
#include <iostream>

using namespace std;

int main(){
    int a = 10;

    //Declaring a pointer
    int* ptr;

    //Initializing a pointer

    ptr = &a;



}
```

Here ptr is a pointer of type int as it stores the address of the variable a of the same type.

In the above we have also initialized the pointer with the address of a which is figured out using the reference operator.

We can use the dereference operator to bring out the value of the variable whose address the pointer has stored. For eg:

```cpp
#include <iostream>

using namespace std;

int main(){
    int a = 10;

    //Declaring a pointer
    int* ptr;

    //Initializing a pointer
    ptr = &a;

    //Using Dereference to get the value
    cout<<"The value of the variable is: "<<*(ptr);
}
```

Dynamic Memory Allocation: Till now whatever codes we wrote had fixed variables who were assigned memory space while the program was compiling. But what if we want to assign memory spaces to pointers only during execution. We do that via Dynamic Memory Allocation. This in itself is a wider topic so only a few important parts are being covered in this note.

Definition: The process of allocating or de-allocating a block of memory during the execution of a program is called Dynamic Memory Allocation. The operators new and delete are utilized for dynamic memory allocation in C++ language, new operator is used to allocate a memory block, and delete operator is used to de-allocate a memory block which is allocated by using new operator.

Let us understand the new and delete keywords using an example:

```cpp
#include <iostream>

using namespace std;

int main(){

    //Using dynamic memory allocation
    int* dtr;
    dtr = new int; //---> Using new keyword to assign memory space to the pointer dtr during runtime
    cout<<"Enter your value: ";
    cin>>*(dtr);   //---> Using the dereference operator to point to the value part of the pointer

    cout<<"The value is: "<<*(dtr);

    delete dtr; //---> Using the delete keyword to delete the pointer once the execution is complete and it frees up memory space.

}
```

The same thing can be done if the pointer is to be assigned to an array.

Assignment: Write a program in c++ to show Dynamic memory allocation using the new and delete keyword for an array of Integers.

Pointers and Arrays:

We know we can store addresses of individual variables into pointers but can it be done for array? Why not. Array is nothing but a combination of several individual elements of the same type.

```
int* aptr;
int arr[5];
aptr = arr;
```

The above snippet shows a pointer aptr storing the address of an array 'arr'. Notice we didn't use &arr and just used arr. This is because the name of any array automatically acts as the address of the first element of the array. So specifying only the name would suffice in this case.

NOTE: This means that aptr currently contains the address of arr[0] and on the use of the dereference operator on it would give the value kept at arr[0].

We can perform increment and decrement operations on the pointer of an array to access the elements of an array. Let us understand this with an example:

```
int main(){
    int* aptr;
    int arr[] = {1,2,3,4,5,6,7,8};
    aptr = arr;
    cout<<"The pointer initially will point to arr[0] which is: "<<*(aptr)<<endl;

    cout<<"But if i want to print the value of the 4th element of the array"<<endl;
    cout<<"The value of arr[3] would be: "<<*(aptr+3)<<endl;


}
```

The above code snippet shows the initial value the pointer points to and how we can print the fourth value of the array by adding 3 to the pointer. Better to recall array indexing.

Functions returning pointers: We have made functions and know that every function has a return type and returns a value. If void it returns nothing, if int it returns integer etc. We can also return pointers. Let us see this in the below example:

```
//Function returning pointer
int* getPtr(){
    int a = 10;
    int* ptr = &a;

    return ptr;
}

int main(){
```

```
    cout<<"The returned pointer value is: "<<*(getPtr());
}
```

In the above code snippet the function getPtr() is returning a pointer of type int and in the main function we use the dereference operator and the returned value to print the value of a. Though doing this in real systems is not good for security its just for learning.

NOTE: We have already covered the reference variable and call by reference parts in previous classes, so refer to that note and if any confusion arises let me know.

Self-Referential Structures:

Definition: The self-referential structure is a structure that points to the same type of structure. It contains one or more pointers that ultimately point to the same structure. They play an important role in data structures such as linked list, trees, graphs etc.

Arrow Operator:

Arrow operator (->) in C++ also known as Class Member Access Operator is used to access the public members of a class, structure, or members of union with the help of a pointer variable.

THIS MARKS THE END OF UNIT – 1 OF YOUR SYLLABUS.