

Chapter 3 :

Token ,Expression and Control Structure

Reference : E. Balaguruswamy Object Oriented Programming With C++; chapter-3.

Topics:

- **Topic 1: Data type,Variable,Operator.**
- **Topic 2: Dynamic memory allocation**

Questions:

Topic 1: Data type,Variable,Operator.

1. What is bool data type? Why array is called derived data type?
2. Describe different types of operators used in C++ and state their purpose.
3. Explain the use of unary operator and ternary operator.
4. Explain the role of scope resolution operator in C++ with an example.
5. What is the reference variable? What is its major use?
6. How does the following statement differ:
i) `char *const p;` ii) `char const *p;`
7. Difference between reference and pointer variable.

Topic 2: Dynamic memory allocation

8. What is dynamic memory allocation? Describe dynamic memory allocation and deallocation operator. Or, How can memory be allocated using “new” and release it using “delete”? Or, Describe the memory management operator used in C++.
Or, What is dynamic initialization of a variable? Give an example.
9. What are the advantages of new operator over malloc() function?
10. Difference between the process of dynamic memory allocation used in C and C++.

Q:1::What is bool(boolean) data type?Why array called derived data type?

Answer:

Bool datatype: A boolean data type is declared with the bool keyword and can only take the values true or false.When the value is returned, true = 1 and false = 0.

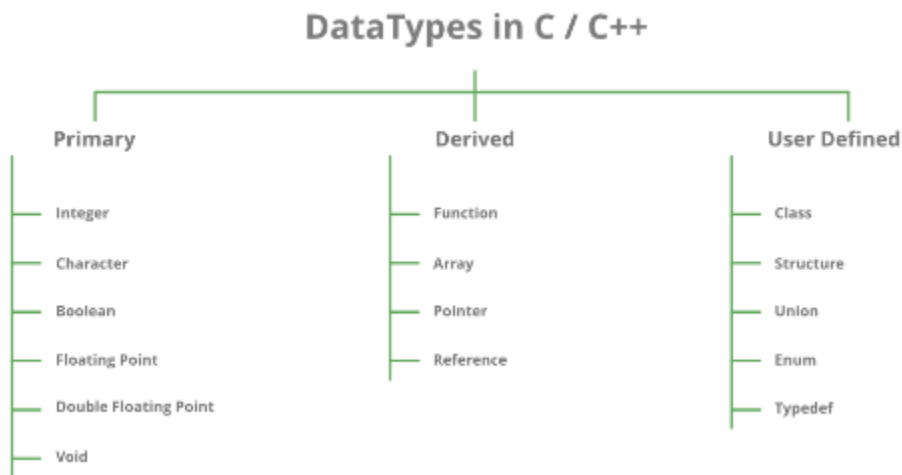
```
#include <iostream>
using namespace std;

int main()
{
    bool a = true;
    bool b = false;
    cout <<"boolean a = "<< a << endl;
    cout <<"boolean b = "<< b;

    return 0;
}
```

```
boolean a = 1
boolean b = 0
```

Array - derived data type :

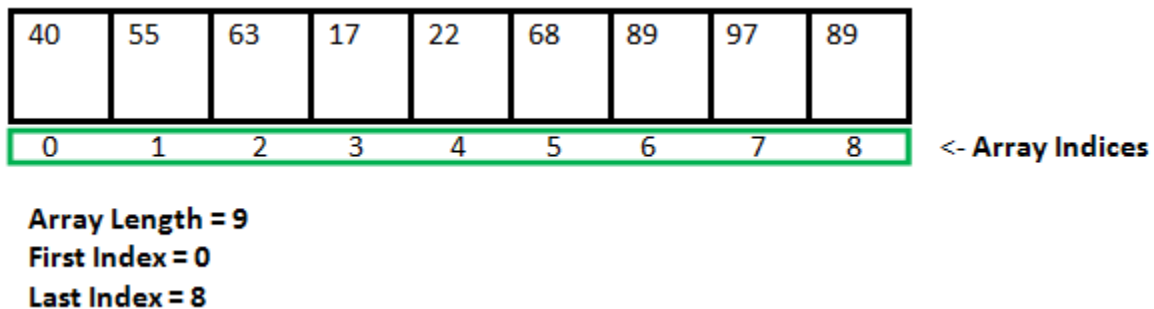


The data-types that are derived from the primitive or built-in datatypes are referred to as derived data types. An array is a collection of elements of a primitive or built-in datatype

where elements are stored at continuous(contiguous) memory locations and represent the elements by one variable .

For example : `int arr[9] = {40,55,63,17,22,68,89,97,89};`

Here arr is an integer type(built in data type) array variable(derived data type) which length is 9 i.e it can store nine integer type elements in a continuous memory space.



That's why an array is called a derived data type. Character type array is called string.

Q:2:: Describe different types of operators used in C++ and state their purpose.

Answer :

Operators can be defined as basic symbols that help us work on logical and mathematical operations. Operators in C++, are tools or symbols that are used to perform mathematical operations concerning arithmetic, logical, conditional and, bitwise operations.

1. Arithmetic Operators

It includes basic arithmetic operations like addition, subtraction, multiplication, division, modulus operations, increment, and decrement. The Arithmetic Operators in C++ include:

1. + (Addition) – This operator is used to add two operands.
2. – (Subtraction) – Subtract two operands.
3. * (Multiplication) – Multiply two operands.
4. / (Division) – Divide two operands and gives the quotient as the answer.
5. % (Modulus operation) – Find the remains of two integers and gives the remainder after the division.
6. ++ (Increment) – Used to increment an operand.
7. — (Decrement) – Used to decrement an operand.

2. Relational Operators

It is used to compare two numbers by checking whether they are equal or not, less than, less than or equal to, greater than, greater than or equal to.

1. == (Equal to)– This operator is used to check if both operands are equal.
2. != (Not equal to)– Can check if both operands are not equal.
3. > (Greater than)– Can check if the first operand is greater than the second.
4. < (Less than)– Can check if the first operand is lesser than the second.
5. >= (Greater than equal to)– Check if the first operand is greater than or equal to the second.
6. <= (Less than equal to)– Check if the first operand is lesser than or equal to the second

If the relational statement is satisfied (it is true), then the program will return the value 1, otherwise, if the relational statement is not satisfied (it is false), the program will return the value 0.

3. Logical Operators

It refers to the boolean values which can be expressed as:

- Binary logical operations, which involves two variables: AND and OR
- Unary logical operation: NOT

Logical Operators in C++ includes –

1. && (AND) – It is used to check if both the operands are true.
2. || (OR) – These operators are used to check if at least one of the operand is true.
3. ! (NOT) – Used to check if the operand is false

If the logical statement is satisfied (it is true), then the program will return the value 1, otherwise, if the relational statement is not satisfied (it is false), the program will return the value 0.

4. Assignment Operators

It is used to assign a particular value to a variable. We will discuss it in detail in the later section with its shorthand notations.

1. = (Assignment)- Used to assign a value from right side operand to left side operand.

2. += (Addition Assignment)- To store the sum of both the operands to the left side operand.
3. -= (Subtraction Assignment) – To store the difference of both the operands to the left side operand.
4. *= (Multiplication Assignment) – To store the product of both the operands to the left side operand.
5. /= (Division Assignment) – To store the division of both the operands to the left side operand.
6. %= (Remainder Assignment) – To store the remainder of both the operands to the left side operand.

5. Bitwise Operators

It is based on the principle of performing operations bit by bit which is based on boolean algebra. It increases the processing speed and hence the efficiency of the program. Bitwise operators are not applicable in the case of float and double. The Bitwise Operators in C++ Includes –

1. & (Bitwise AND) – Converts the value of both the operands into binary form and performs AND operation bit by bit.
2. | (Bitwise OR) – Converts the value of both the operands into binary form and performs OR operation bit by bit.
3. ^ (Bitwise exclusive OR) – Converts the value of both the operands into binary form and performs EXCLUSIVE OR operation bit by bit.
4. ~ (One's complement operator): Converts the operand into its complementary form.
5. << – Left shift
6. >> – Right shift

6. Miscellaneous Operators Apart from the above-discussed operators, there are certain operators which fall under this category which include sizeof and ternary (conditional) operators. Here is a list which illustrates the use of these operators:

1. sizeof – It returns the memory occupied by the particular data type of the operand
2. & (reference or address) – It refers to the address (memory location) in which the operand is stored.
3. * (Pointer) – It is a pointer operator
4. ? (Condition) – It is an alternative for if-else condition

Q:3:: Explain the use of unary operator and ternary operator.

Answer :

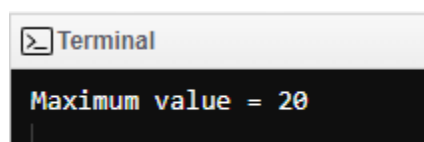
The three categories of operators based on the number of operands they require are:

1. **Unary** : A unary operator is an operator, which operates on one operand. The sizeof, increment, decrement, unary- are unary operators. Example :

```
#include <iostream>
using namespace std;
int main()
{
    int a = 8, b=10;
    a++;
    --b;
    cout<<"a = "<<a<<endl;    //a=9
    cout<<"b = "<<b<<endl;    //b=9
    return 0;
}
```

2. **Binary** : A binary operator is an operator, which operates on two operands. Arithmetic, logical, relational, assignment, bitwise-operators are binary operators.
3. **Ternary** : A ternary operator is an operator, which operates on three operands. There is one ternary operator(? :). The ternary operator is also called a "Conditional Operator". Example :

```
1  #include <iostream>
2  using namespace std;
3  int main()
4  {
5      int a = 10;
6      int b = 20;
7      int max = a > b ? a : b;
8      cout << "Maximum value = " << max << "\n";
9      return 0;
10 }
11
```



Terminal

Maximum value = 20

Q:4:: Explain the role of scope resolution operator in C++ with an example.

Answer:

The double colon (::) symbol is known as scope resolution operator or scope operator in C++ programming language.

Role of the scope resolution operator

1. It is used to access the member variables or member functions of a program.
2. It defines the member function outside of the class using the scope resolution.
3. It is used to access the static variable and static function of a class.
4. The scope resolution operator is used to override function in the Inheritance.
5. The scope resolution operator used to access the global variable from a function. For example- In the following program the variable m is declared at three places- outside the main(), inside main() function but outside the inner block and inside an inner block.

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

int m = 10;
int p;
int main()
{
    int m = 20;
    {
        int k = m;
        int m = 30;
        cout<<"From inner block the inner block m = "<<m<<endl;
        cout<<"From inner block the global m(::m) = "<<::m<<endl;
        cout<<"From inner block the outside of inner block m(=k) = "<<k<<endl<<endl;
        p = m;
    }
    cout<<"From outside of inner block the outside of inner block m = "<<m<<endl;
    cout<<"From outside of inner block the global m(::m) = "<<::m<<endl;
    cout<<"From outside of inner block the inside of inner block m = "<<p<<endl;

    return 0;
}
```

```
From inner block the inner block m = 30
From inner block the global m(::m) = 10
From inner block the outside of inner block m(=k) = 20

From outside of inner block the outside of inner block m = 20
From outside of inner block the global m(::m) = 10
From outside of inner block the inside of inner block m = 30

Process returned 0 (0x0)   execution time : 0.047 s
Press any key to continue.
```

Github code -

https://github.com/SyedaJannatul/DIIT_Object-Oriented-Programming/blob/bf2e0fa0ec3df5f47bf00eaa3d8c743114b0bed5/chapter%20_token%2Cexpression%20and%20control%20structures/chapter%20_class_code/question%204_code.cpp

Q:5:: What is the reference variable? What is its major use?

Answer:

The operator '&' is used to declare a reference variable. Reference variable is an alternate name of an already existing variable. It cannot be changed to refer to another variable and should be initialized at the time of declaration and cannot be NULL.

Reference variable is nothing but just giving a different name to a pre-existing variable. It is used mainly in function-call when we use call by reference or pass by reference technique to pass the origin variable to the function instead of pass copy of the variable.

Here is an example of reference variable:


```

#include <iostream>
using namespace std;
int main()
{
    int a = 8;
    int &b = a;
    int c = a;

    cout << "The variable a value : " << a << endl;
    cout << "The reference variable b value : " << b << endl;
    cout << "The variable c value : " << c << endl<< endl;

    cout << "The &a : " << &a << endl<< endl;
    cout<<"&b = a"<<endl;
    cout << "The &b : " << &b << endl<< endl;
    cout<<"c = a"<<endl;
    cout << "The &c : " << &c << endl;

    return 0;
}

```

```

The variable a value : 8
The reference variable b value : 8
The variable c value : 8

The &a : 0x61fe14

&b = a
The &b : 0x61fe14

c = a
The &c : 0x61fe10

Process returned 0 (0x0)   execution time : 0.057 s
Press any key to continue.

```

Q:6:: How does the following statement differ:

- i) `char *const p;`
- ii) `char const *p;`

Answer :

char *const ptr	char const *ptr or, const char *ptr	const char *const ptr
“char * const ptr” is a constant pointer to a character. This means that the pointer cannot be changed.	“const char *ptr” is a pointer to a constant character. This means that the value stored in the pointer cannot be changed.	“const char *const ptr” is a constant pointer to a constant character.
The character data stored in the pointer can be changed but the pointer cannot be changed.	The pointer can be used to access the character data, but the data itself cannot be modified.	We can neither change the value pointed by the pointer variable ptr nor the pointer ptr.
This type of pointer is useful for storing strings of characters that may need to be modified.	This type of pointer is useful for storing strings of characters that will not be changed or that may need to be accessed in different ways or that may need to be accessed in different ways.	We cannot change the value of the pointer as well as it is now constant and it cannot point to another constant char.
This is a constant pointer to non constant character	This is a non-constant pointer to constant character	This is a constant pointer to constant character.

Here is an example :

Github code-

https://github.com/SyedaJannatul/DIIT_Object-Oriented-Programming/blob/bf2e0fa0ec3df5f47bf00eaa3d8c743114b0bed5/chapter%20_token%2Cexpression%20and%20control%20structures/chapter%203_class_code/question%206_code.cpp

```

#include <iostream>
using namespace std;
int main()
{
    char a ='A', b ='B';
    //
    const char *ptr;
    ptr = &a;
    cout<<"ptr = &a"<<endl;
    cout<<"value pointed by ptr = "<<*ptr<<endl;
    ptr = &b; //ptr = b is illegal here,because it trying to
              //change the character value.
    cout<<"ptr = &b"<<endl;
    cout<<"value pointed by ptr = "<<*ptr<<endl;
    //
    char *const pt = &a;    //pt need to be initialized here.
    cout<<"pt = &a"<<endl;
    cout<<"value pointed by pt = "<<*pt<<endl;
    *pt = b;    //ptr = &b is illegal here,because it trying to
                //change the pointer value.
    cout<<"*pt = b"<<endl;
    cout<<"value pointed by pt = "<<*pt<<endl;
    //
    const char *const p = &a;
    cout<<"p = &a"<<endl;
    cout<<"value pointed by p = "<<*p<<endl;
    //p=&b is illegal
    //*p=b,is illegal
    return 0;
}

```

```

ptr = &a
value pointed by ptr = A
ptr = &b
value pointed by ptr = B
pt = &a
value pointed by pt = A
*pt = b
value pointed by pt = B
p = &a
value pointed by p = B

```

```

Process returned 0 (0x0)   execution time : 0.063 s
Press any key to continue.

```

Q:7::Difference between reference and pointer variable.

Answer:

	Reference variable		Pointer variable
1.	At the time of reference variable creation it needs to be initialized, otherwise it generates errors.	1.	At the time of pointer variable creation it is not mandatory to initialize it.
2.	The concept of NULL reference is not allowed.	2.	The concept of NULL reference is allowed.
3.	Reference variable is nothing but just giving a different name to a pre-existing variable.	3.	Pointer is a special type variable which stores the address of a pre-existing variable and points to the value of that variable.

```
#include <iostream>
using namespace std;
int main()
{
    int a = 8;
    int &b = a;
    int c = a;
    int *d;
    d = &a;

    cout << "The variable a value : " << a << endl;
    cout << "The reference variable b value : " << b << endl;
    cout << "The variable c value : " << c << endl<< endl;
    cout << "The pointer variable *d value : " << *d << endl;
    cout << "The pointer variable d value : " << d << endl<< endl;

    cout << "The &a : " << &a << endl<< endl;
    cout<<"&b = a"<<endl;
    cout << "The &b : " << &b << endl<< endl;
    cout<<"c = a"<<endl;
    cout << "The &c : " << &c << endl<< endl;
    cout<<"*d = &a"<<endl;
    cout << "The &d : " << &d << endl;

    return 0;
}
```

```
The variable a value : 8
The reference variable b value : 8
The variable c value : 8

The pointer variable *d value : 8
The pointer variable d value : 0x61fe14

The &a : 0x61fe14

&b = a
The &b : 0x61fe14

c = a
The &c : 0x61fe10

*d = &a
The &d : 0x61fe08

Process returned 0 (0x0)   execution time : 0.098 s
Press any key to continue.
```

Github code -

https://github.com/SyedaJannatul/DIIT_Object-Oriented-Programming/blob/bf2e0fa0ec3df5f47bf00eaa3d8c743114b0bed5/chapter%203_token%20expression%20and%20control%20structures/chapter%203_class_code/question%207_code.cpp

Q:8:: What is dynamic memory allocation? Describe dynamic memory allocation and deallocation operator.

Or, How can memory be allocated using “new” and release it using “delete”?

Or, Describe the memory management operator used in C++.

Or, What is dynamic initialization of a variable? Give an example.

Answer:

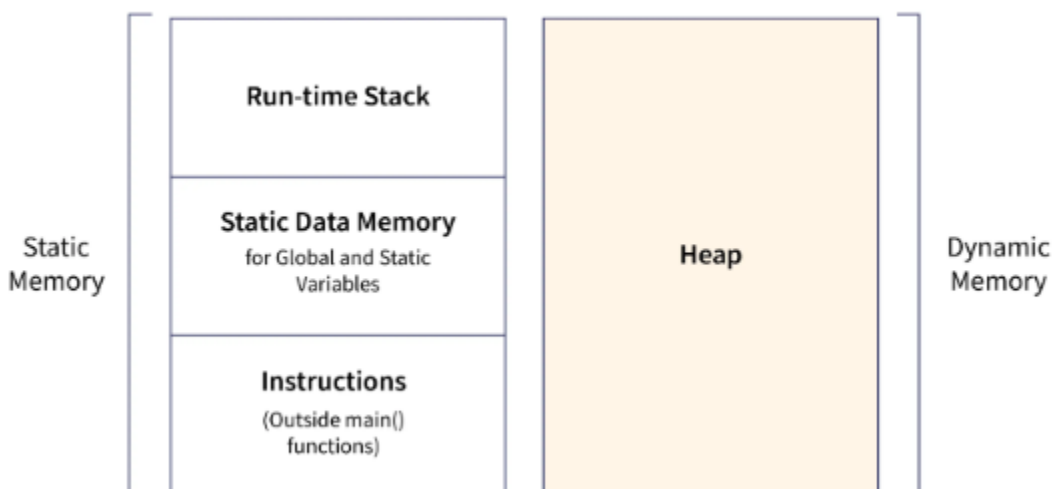
Memory allocation : Reserving or providing space to a variable is called memory allocation. For storing the data, memory allocation can be done in two ways -

- **Static allocation or compile-time allocation** - Static memory allocation means providing space for the variable. The size and data type of the variable is known, and it remains constant throughout the program.
- **Dynamic allocation or run-time allocation** - The allocation in which memory is allocated dynamically. In this type of allocation, the exact size of the variable is not known in advance. Pointers play a major role in dynamic memory allocation. Dynamically we can allocate storage while the program is in a running state, but variables cannot be created "on the fly". Thus, there are two criteria for dynamic memory allocation -
 - A dynamic space in the memory is needed.
 - Storing the address to access the variable from the memory

Similarly, we do memory de-allocation for the variables in the memory.

In C++, memory is divided into two parts -

- **Stack** - All the variables that are declared inside any function take memory from the stack.
- **Heap** - It is unused memory in the program that is generally used for dynamic memory allocation.



Dynamic memory allocation operator and deallocation operator/memory management operator :

In C++ Language, new and delete operators are pre-defined in the C++ Standard Library and don't require to include any library file for run-time allocation and deallocation of memory blocks. Although we can use malloc(), calloc(), and free() functions of C language in C++ as well by adding the <stdlib.h> header file because of the backward compatibility of C++ with C Language.

The new and delete operators are known as dynamic memory allocation operators or memory allocation operators or free store operators(since these operators manipulate the memory on free store).

The "new" operator in C++ :

To allocate the space dynamically, the operator new is used. It means creating a request for memory allocation on the free store(heap). If memory is available, memory is initialized and the address of that space is returned to a pointer variable. Syntax for "new" operator:

```
data_type    *ptr_var = new    data_type; or
data_type    *ptr_var = new    data_type(value);
```

Here ,

- ☐ ptr_var is a pointer, which stores the address of the type data_type.
- ☐ Any pre-defined data types, like int, char, etc., or other user-defined data types, like classes, can be used as the data_type with the new operator.
- ☐ Value is assigned value to memory by pointer.

The "delete" Operator in C++

The delete operator is used to de-allocate the block of memory, which is dynamically allocated using the new operator. Since, a programmer must de-allocate a block of memory, once it is not required in the program. So, we have to use the delete operator to avoid memory leaks and program crash errors, which occur due to the exhaustion of the system's memory. Syntax to delete a single block of memory: delete ptr_var;

Dynamic initialization of a variable:

Dynamic initialization of a variable method : 1

```

#include <iostream>
using namespace std;
int main()
{
    // 1.dynamically allocating an integer memory block
    int *ptr = new int;
    // storing a value at the memory pointed by ptr
    *ptr = 8;
    cout << "Value at memory pointed by ptr= " << *ptr<<endl;
    // ptr memory deallocated using the delete operator
    delete ptr;
    cout<<"Garbage value: "<<*ptr<<endl;
    ptr = NULL;
    cout<<"*ptr value: "<<*ptr<<endl;
    return 0;
}

```

```

Value at memory pointed by ptr= 8
Garbage value: 7084592
*ptr value:
Process returned -1073741819 (0xC0000005)   execution time : 0.625 s
Press any key to continue.

```

Dynamic initialization of a variable method : 2

```

#include <iostream>
using namespace std;
int main()
{
    // 2.dynamically allocating an integer memory block
    // with 100 as its initial value
    int *p = new int(100);
    cout << "Value at memory pointed by p= " << *p<<endl;
    // p memory deallocated using the delete operator
    delete p;
    cout<<"Garbage value: "<<*p<<endl;
    p = NULL;
    cout<<"*p value: "<<*p;

    return 0;
}

```

```

Value at memory pointed by p= 100
Garbage value: 16063024
*p value:
Process returned -1073741819 (0xC0000005)   execution time : 0.625 s
Press any key to continue.

```


Dynamic initialization of a variable method : 3

```
#include <iostream>
using namespace std;
int main()
{
    /*3.When there is insufficient memory in the heap segment
    during the run-time of a C++ program, the new request
    for allocation fails by throwing an exception of type
    std::bad_alloc. It can be avoided using a nothrow argument
    with the new operator. When we use nothrow with new, it returns
    a NULL pointer. So, the pointer variable should be checked that
    is formed by new before utilizing it in a program.*/
    int *pt = new(nothrow) int;
    if (pt == NULL)
    {
        cout << "Allocation Failed!\n";
    }
    *pt = 502;
    cout << "Value at memory pointed by pt= " << *pt<<endl;
    // pt memory deallocated using the delete operator
    delete pt;
    cout<<"Garbage value: "<<*pt<<endl;
    pt = NULL;
    cout<<"*pt value: "<<*pt;
    return 0;
}
```

```
Value at memory pointed by pt= 502
Garbage value: 7543344
*pt value:
Process returned -1073741819 (0xC0000005)   execution time : 0.750 s
Press any key to continue.
```

Q:9:: What are the advantages of new operator over malloc() function?

Answer :

The new operator offers the following advantages over the function malloc().

1. It automatically computes the size of the data object . We need not use the operator sizeof.
2. It automatically returns the correct pointer type, so that there is no need to use a type cast.
3. It is possible to initialize the object while creating the memory space.
4. like any other operator, new and delete can be overloaded.

Q:10::Difference between the process of dynamic memory allocation used in C and C++.

Answer :

	Dynamic memory allocation in C++ language	Dynamic memory allocation in C language
Library used for dynamic memory allocation	<code>#include<iostream></code> using namespace std;	<code>#include<stdlib.h></code>
Memory allocation for default data types	new operator used.Syntax: data_type *ptr_var = new data_type; Example: creating array: int *ptr = new int(10);	malloc() function used.Syntax: data_type *pointer_variable = (cast_type*)malloc(byte-size); Example:creating array: int *ptr; ptr = (int*)malloc(10*sizeof(int));
Memory allocation for derived data types	new operator used.	calloc() function used
Free the previously allocated memory	delete operator used.Syntax: delete pointer_variable;	free() function used.Syntax: free(pointer_variable);
Modified the size of the previously allocated memory	There is no explicit operator or function to perform this operation.	realloc() function used.
Check memory space is available or not	nothrow argument with the new operator used.Example: int *pt = new(nothrow) int; if (pt == NULL) { cout <<"AllocationFailed!\n"; }	There is no explicit operator or function to perform this operation.

