DYNAMIC MEMORY ALLOCATION

Memory allocation in C++ can be done in three ways:

1. Automatic Memory Allocation
2. Static Memory Allocation
3. Dynamic Memory Allocation

Automatic Memory allocation is done when we declare variables or objects in our program. Memory in your C++ program is divided into two parts:

- The stack: All variables declared inside the function will take up memory from the stack.

- The heap: This is unused memory of the program and can be used to allocate the memory dynamically when program runs.

Many times, you are not aware in advance how much memory you will need to store particular information in a defined variable and the size of required memory can be determined at run time.
You can allocate memory at run time within the heap for the variable of a given type using a special operator in C++ which returns the address of the space allocated. This operator is called new operator.
If you are not in need of dynamically allocated memory anymore, you can use delete operator, which de-allocates memory previously allocated by new operator.

1. Automatic memory allocation:
   Suppose you create a function print( ). In this function, you are declaring a variable of integer type. When this function terminates, the memory allocated to that variable automatically gets de-allocated by the compiler. This is called as automatic memory allocation.
   ```
   void print( )
   {
           int a;  //Memory is allocated to variable a automatically by the compiler
           a = 10;
           cout<<"Printing the value of a:";
           cout<<a; //prints the value of a as 10
   } //function print( ) terminates.

   cout<<a;  //error: a is not declared in this scope.
   ```
   As soon as the function terminates, the variable gets destroyed automatically by the compiler because the scope of the variable was only within the function block.

2. Static memory allocation:

As we know that the lifetime of the static variable is within the whole program. You have learned this concept in Unit 2nd. When the variables are declared as static, this is called as static memory allocation.

3. Dynamic Memory Allocation
   In both Automatic and Static memory allocation, memory was allocated to variables during the compilation time i.e. when you compile your program. Many times, you are not aware in advance how much memory you will need to store particular information in a defined variable and the size of required memory can be determined at run time.
   C++ provides two dynamic allocation operators:
   new and delete. These operators are used to allocate and free memory at run time. Dynamic allocation is an important part of almost all real-world programs. As explained in Part One, C++ also supports dynamic memory allocation functions, called malloc() and free(). These are included for the sake of compatibility with C. However, for C++ code, you should use the new and delete operators because they have several advantages. The new operator allocates memory and returns a pointer to the start of it. The delete operator frees memory previously allocated using new.
   You can allocate memory at run time within the heap for the variable of a given type using a special operator in C++ which returns the address of the space allocated. This operator is called new operator.
   For example,

```
int  *ptr = new int;
```

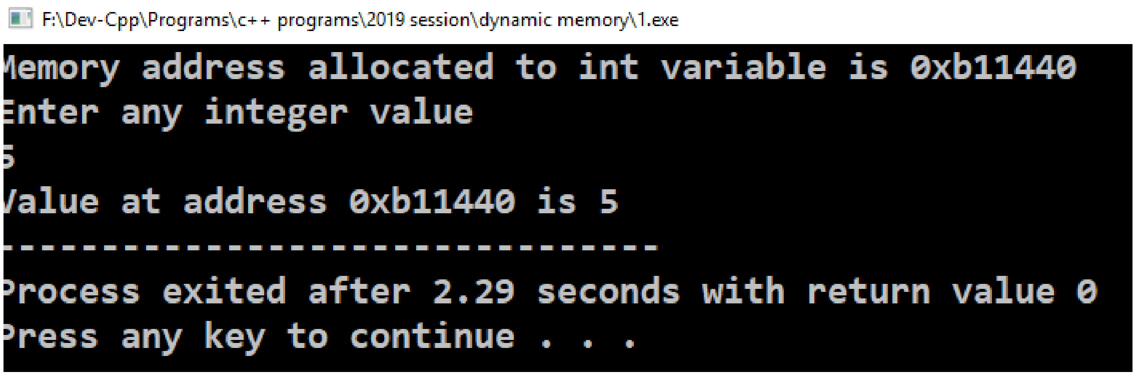
   statement will allocate memory to an integer type of variable during runtime. Please note that dynamic type of data don't have any name here.
   Similarly, you can write..

```
float *ptr = new float;
 char *ptr = new char;
Student *ptr = new Student;
```

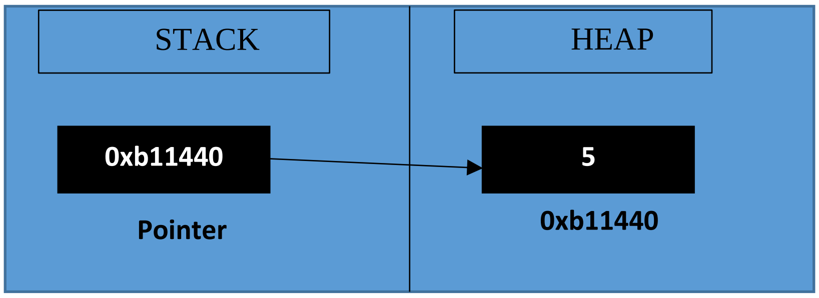
   where Student is class name and new create dynamic object without any name.
   If you are not in need of dynamically allocated memory anymore, you can use delete operator, which de-allocates memory previously allocated by new operator.

PROGRAM 1. Let's see first program to allocate memory during runtime to store integer value

```
#include<iostream>
using namespace std;
int main()
{
        /*new operator would return memory address from heap
        randomly. Therefore, you would get different memory address
        for every program execution. */
        int *ptr= new int; //allocating memory

        cout<<"Memory address allocated to int variable is "<<ptr;
        cout<<"\nEnter any integer value"<<endl;
        cin>>*ptr;

        cout<<"Value at address "<<ptr<<" is "<<*ptr;
        delete ptr;//deallocating memory
        return 0;
}
```

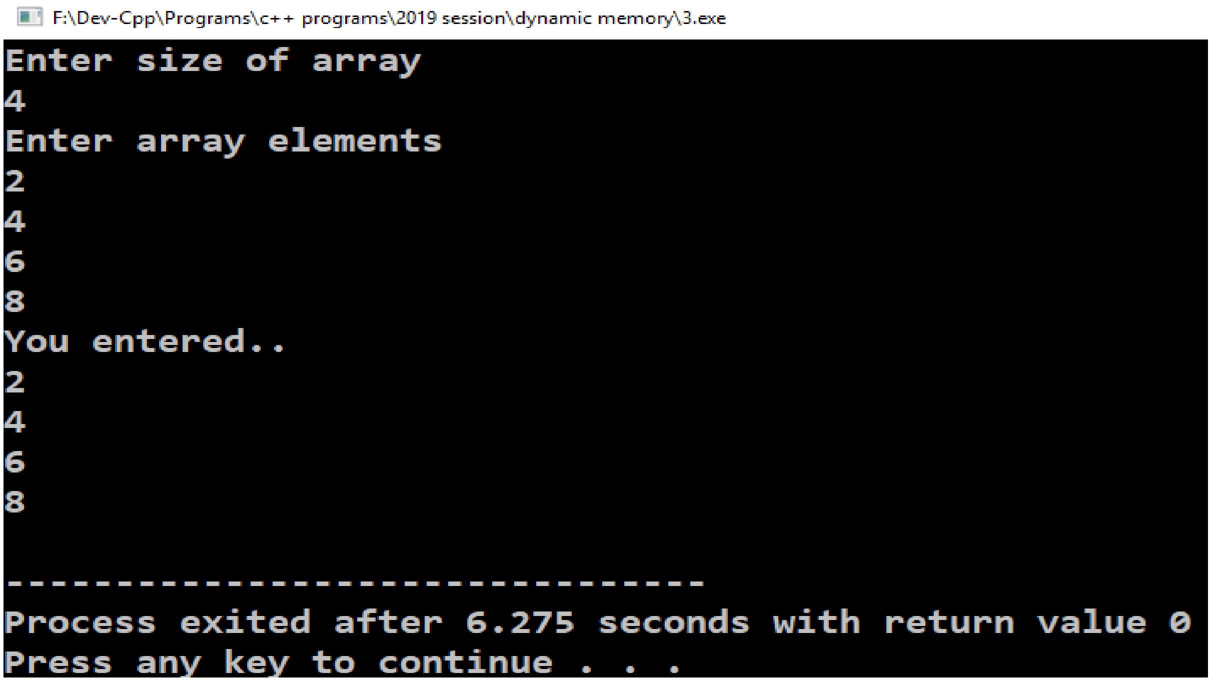


```
F:\Dev-Cpp\Programs\c++ programs\2019 session\dynamic memory\1.exe
Memory address allocated to int variable is 0xb11440
Enter any integer value
5
Value at address 0xb11440 is 5
--------------------------------
Process exited after 2.29 seconds with return value 0
Press any key to continue . . .
```



PROGRAM 2. To allocate memory dynamically to 1-D array

```
#include<iostream>
using namespace std;
int main()
{
        int i,x;
        cout<<"Enter size of array\n";
        cin>>x;
        int *ptr= new int[x]; //allocating memory

    cout<<"Enter array elements\n";
        for(i=0;i<x;i++)
        cin>>*(ptr+i);

        cout<<"You entered..\n";
        for(i=0;i<x;i++)
        cout<<*(ptr+i)<<endl;
        delete[] ptr;//deallocating memory
        return 0;
}
```



F:\Dev-Cpp\Programs\c++ programs\2019 session\dynamic memory\3.exe

```
Enter size of array
4
Enter array elements
2
4
6
8
You entered..
2
4
6
8

--------------------------------
Process exited after 6.275 seconds with return value 0
Press any key to continue . . .
```

Program 3 to allocate memory dynamically to a 2-D array.

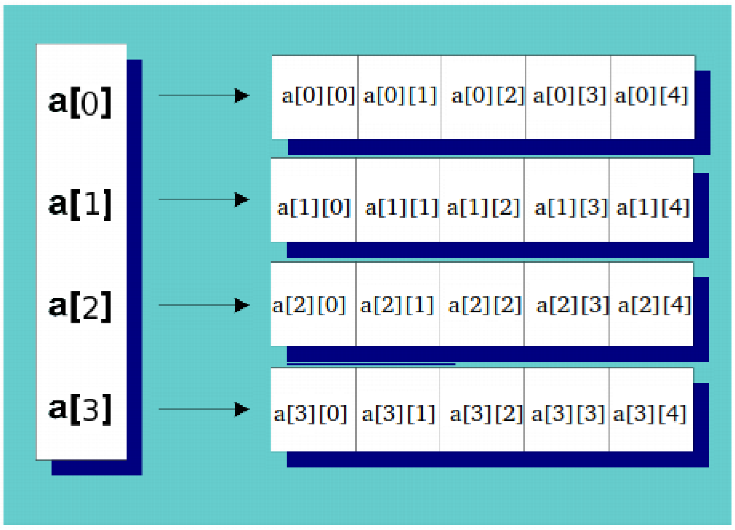
A dynamic 2D array is basically an array of pointers to arrays. You can initialize it using a loop, like this:

*int** a = new int*[rowCount];*

for(int i = 0; i < rowCount; ++i)

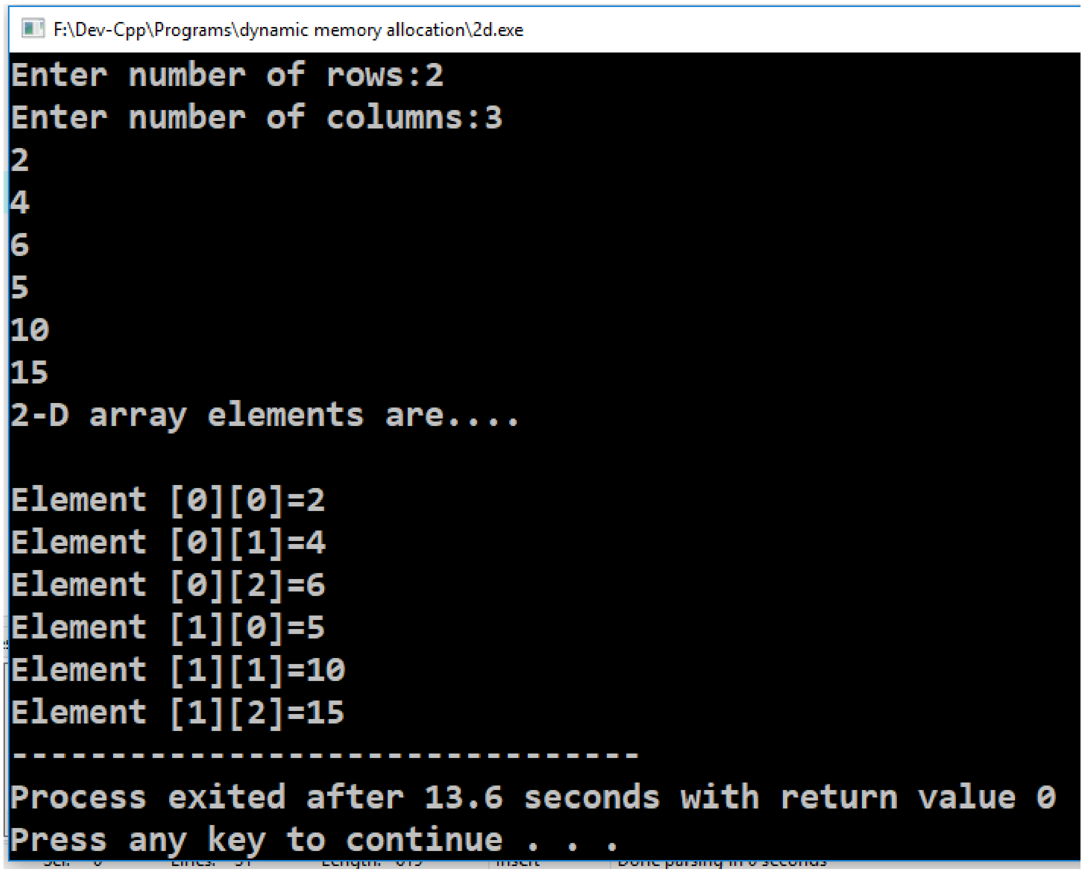
a[i] = new int[colCount];

The above, for colCount= 5 and rowCount = 4, would produce the following:


a[0]
a[1]
a[2]
a[3]
a[0][0] a[0][1] a[0][2] a[0][3] a[0][4]
a[1][0] a[1][1] a[1][2] a[1][3] a[1][4]
a[2][0] a[2][1] a[2][2] a[2][3] a[2][4]
a[3][0] a[3][1] a[3][2] a[3][3] a[3][4]

```
#include<iostream>
using namespace std;
int main()
{
        int rows,cols;
        cout<<"Enter number of rows:";
        cin>>rows;
        cout<<"Enter number of columns:";
        cin>>cols;
        //creating a dynamic array of pointers which will further point to 1-D arrays
        int** ptr = new int*[rows];
  for(int i = 0; i < rows; ++i)
        ptr[i] = new int[cols];
        for(int i=0;i<rows;i++)
        {
                for(int j=0;j<cols;j++)
                {
                        cin>>ptr[i][j];
                }
        }
        cout<<"2-D array elements are....\n";
        for(int i=0;i<rows;i++)
        {
                for(int j=0;j<cols;j++)
                {
                        cout<<"\nElement ["<<i<<"]["<<j<<"]="<<ptr[i][j];
                }
        }
        return 0;
```

}



F:\Dev-Cpp\Programs\dynamic memory allocation\2d.exe

```
Enter number of rows:2
Enter number of columns:3
2
4
6
5
10
15
2-D array elements are....

Element [0][0]=2
Element [0][1]=4
Element [0][2]=6
Element [1][0]=5
Element [1][1]=10
Element [1][2]=15
--------------------------------
Process exited after 13.6 seconds with return value 0
Press any key to continue . . .
```

<u>Allocating memory dynamically to an object</u>

WE CAN CREATE OBJECTS IN 2 WAYS:

- Creating Non-dynamic objects (i.e. automatic)
  Example Student s1;
  where Student is class name and s1 is the object name.
- Creating dynamic object
  Example Student *ptr = new Student;
  where Student is the class name and ptr is the pointer variable

Here, we are creating a pointer to un-named object. We can achieve dynamic memory allocation only using pointers.

✓ When you write, Student *ptr, it means you are creating a pointer of type object(which kind of object? Object of type Student. Just like you create pointer of type integer by writing int *ptr).
✓ new is a keyword in C++.
✓ new operator is used to dynamically allocate memory to variable. Here,
  new Student
  means that we are telling the compiler to reserve some memory randomly for the object of type Student.

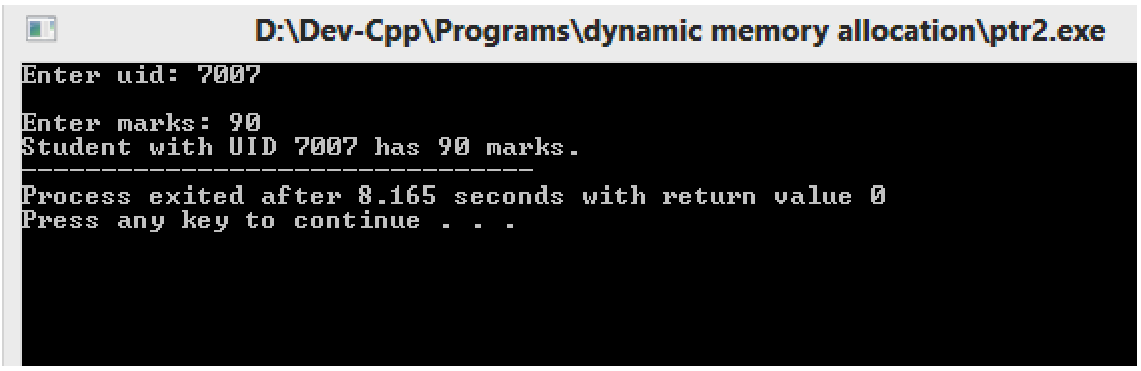
- ✓ new operator returns the address of unnamed object. Since, we don't know how many objects we want to create. We will be telling it at run time. This is explained in Program 5.
- ✓ The address returned by new operator (let's say 0013ff60 is the address) is being stored in a pointer variable.
- ✓ We can divide the single statement in two statements as below:
    - Student *ptr ;
    - ptr = new Student ;

- ✓ The memory being allocated to a variable using new operator must be de-allocated by delete operator.
- ✓ new operator calls the default constructor (if no constructor is provided) and returns the address

- ✓ delete operator calls the default destructor (if no destructor is provided).
- ✓ If we create object dynamically i.e. using new operator then that dynamic object can call the private constructor as well as destructor but non-dynamic objects cannot call private constructor and destructor functions.

Let's see the example i.e. our next program.

```
/*Program 4 CREATING DYNAMIC OBJECT.
In this program, only 1 object has been created dynamically. Dynamic object creation means
we don't know at run time that how many objects we want to create. So, we will be asking user
for the number of objects at run time in next Program 4 */
#include<iostream>
using namespace std;
class student
{
        int uid;
        int marks;
        public:
                void getmarks() //creating member function of class student
                {
                        cout<<"Enter uid: ";
                        cin>>uid;
                        cout<<"\nEnter marks: ";
                        cin>>marks;
                        cout<<"Student with UID "<<uid<<" has "<<marks<<" marks. ";
                }
};
int main()
{
        student *ptr= new student; //creating dynamic object through new operator
        ptr->getmarks(); //accessing the function via pointer through arrow operator
        delete ptr; //deallocating memory through delete operator
        //(*ptr).getmarks(); is equivalent to above line.
        return 0;
}
```



D:\Dev-Cpp\Programs\dynamic memory allocation\ptr2.exe

```
Enter uid: 7007

Enter marks: 90
Student with UID 7007 has 90 marks.
--------------------------------
Process exited after 8.165 seconds with return value 0
Press any key to continue . . .
```

Fig 3. Program 3

Now, in this program we have created only single object.

Now we will create another program, in which we don't know initially how many objects we want to create. We will be allocating memory to the objects at run time i.e. execution time.

For example: int array[10]; //Here, we are telling the size of array at the compile time.

But if we write: int array[x]; //We will get an error.

i.e. we cannot allocate memory to automatic variable at run time i.e. when we run our program. This can be achieved only through new and delete operator.

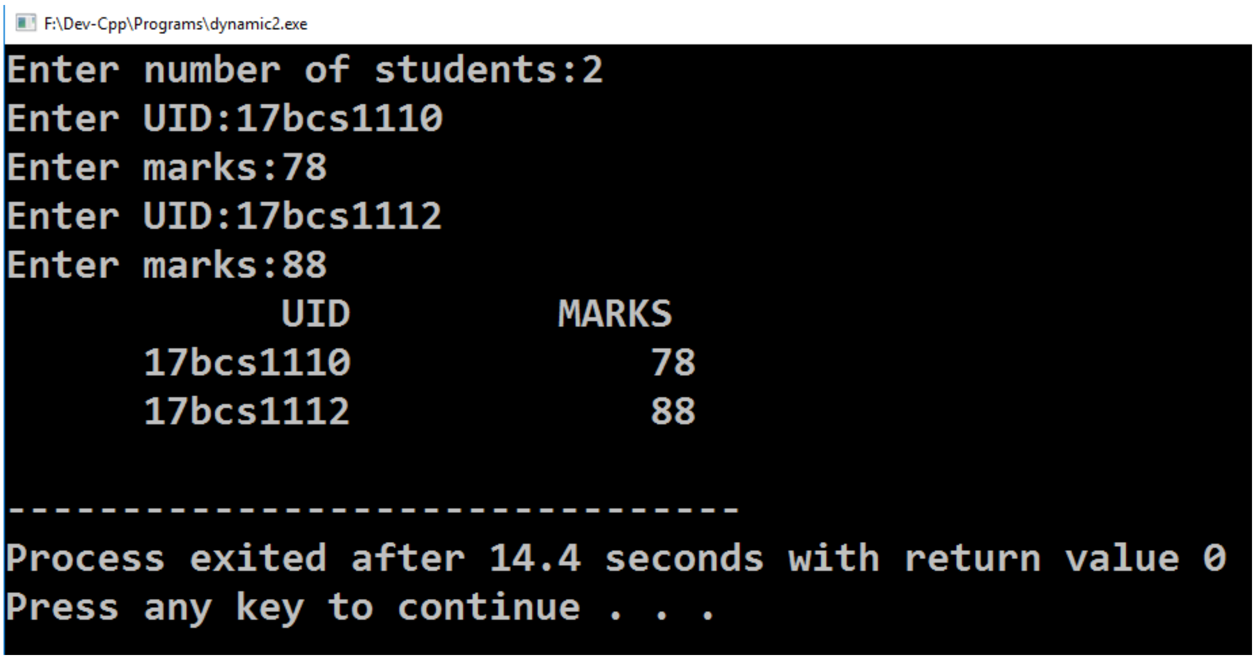
Let's see another Program, where we don't know how many objects we want to create. We will be asking the user that how many record he want to enter. So objects will be created according to the user choice only.

PROGRAM 5...

Creating objects dynamically i.e. allocating memory to objects at run time and asking user that how many objects you want to create at the run -time i.e. when user execute the program.

```
#include<iostream>
#include<iomanip>
#include<stdlib.h>
#include<cctype>
using namespace std;
class student
{
        int uid;
        int marks;
```

```
	public:
		void getmarks()
		{
			cout<<"\nEnter uid: ";
			cin>>uid;
			cout<<"\nEnter marks: ";
			cin>>marks;
			cout<<"\nStudent with UID "<<uid<<" has "<<marks<<" marks.\n ";
		}
};

int main()
{
	int size,i;
	cout<<"Enter number of students:";
	cin>>size;
	if(isspace(size))
	{
		cout<<"You have entered incorrect input..";
		exit(1);
	}
	student *ptr=new student[size];
	for(i=0;i<size;i++)
	{
		(ptr+i)->get();

	}
	cout<<setw(15)<<"UID"<<setw(15)<<"MARKS\n";
	for(i=0;i<size;i++)
	{
		(ptr+i)->show();
	}
	return 0;
}
```



```
F:\Dev-Cpp\Programs\dynamic2.exe
Enter number of students:2
Enter UID:17bcs1110
Enter marks:78
Enter UID:17bcs1112
Enter marks:88
            UID          MARKS
      17bcs1110              78
      17bcs1112              88

--------------------------------
Process exited after 14.4 seconds with return value 0
Press any key to continue . . .
```

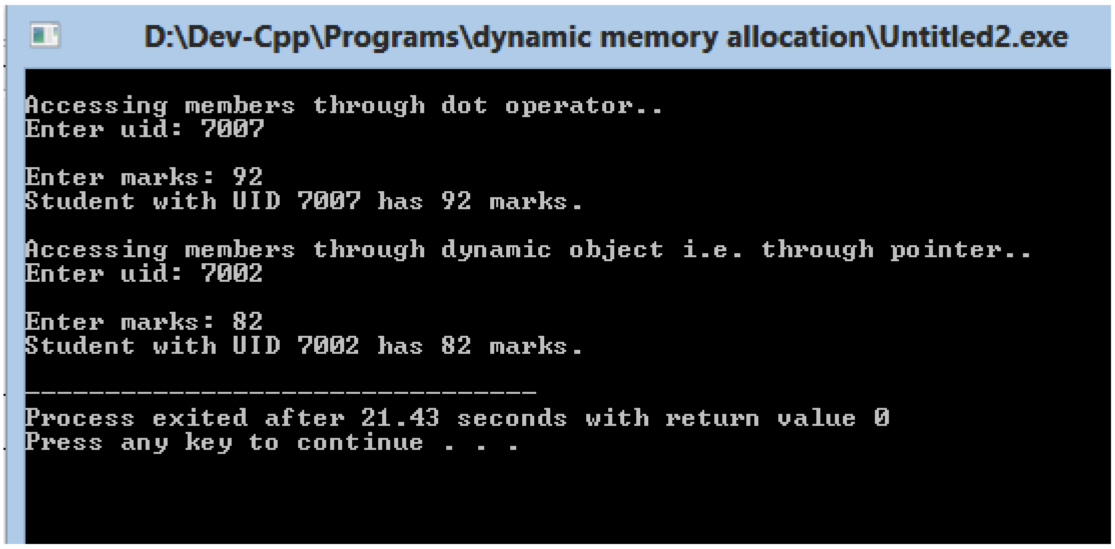
Fig 4. Program 4

PROGRAM 6

Creating dynamic object and non-dynamic object. Output for both will be the same.

```
#include<iostream>
using namespace std;
class student
{
	int uid;
	int marks;
	public:
		void getmarks()
		{
			cout<<"Enter uid: ";
			cin>>uid;
			cout<<"\nEnter marks: ";
			cin>>marks;
			cout<<"Student with UID "<<uid<<" has "<<marks<<" marks.\n ";
		}
};
int main()
{
	student s1; //creating non-dynamic object
	cout<<"\nAccessing members through dot operator..\n";
	s1.getmarks();
	cout<<"\nAccessing members through dynamic object i.e. through pointer..\n";
	student *ptr =new student; //creating dynamic object
	ptr->getmarks();
```

```
        return 0;
}
```

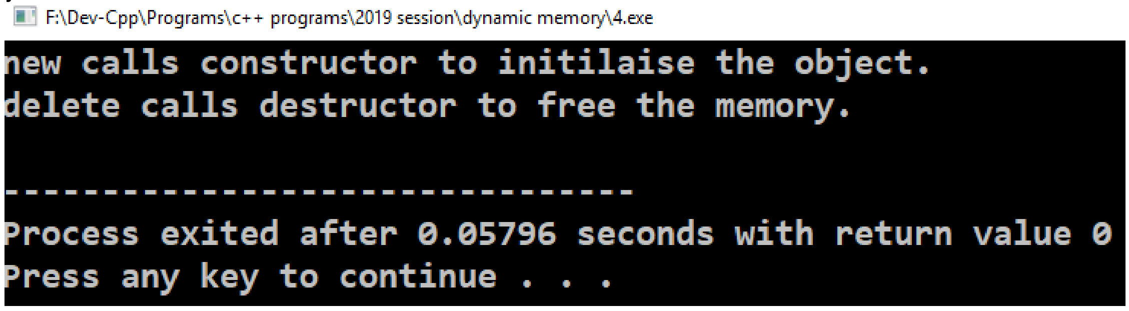


D:\Dev-Cpp\Programs\dynamic memory allocation\Untitled2.exe

```
Accessing members through dot operator..
Enter uid: 7007

Enter marks: 92
Student with UID 7007 has 92 marks.

Accessing members through dynamic object i.e. through pointer..
Enter uid: 7002

Enter marks: 82
Student with UID 7002 has 82 marks.

--------------------------------
Process exited after 21.43 seconds with return value 0
Press any key to continue . . .
```

Program 7 to illustrate that new calls constructor and delete calls destructor to allocate and deallocate memory to dynamic object.

```
#include<iostream>
using namespace std;
class Test
{
        public:
                Test()
                {
                        cout<<"new calls constructor to initilaise the object.\n";
                }
                ~Test()
                {
                        cout<<"delete calls destructor to free the memory.\n";
                }
};
int main()
{
        Test *ptr= new Test;
        delete ptr;
        return 0;
}
```

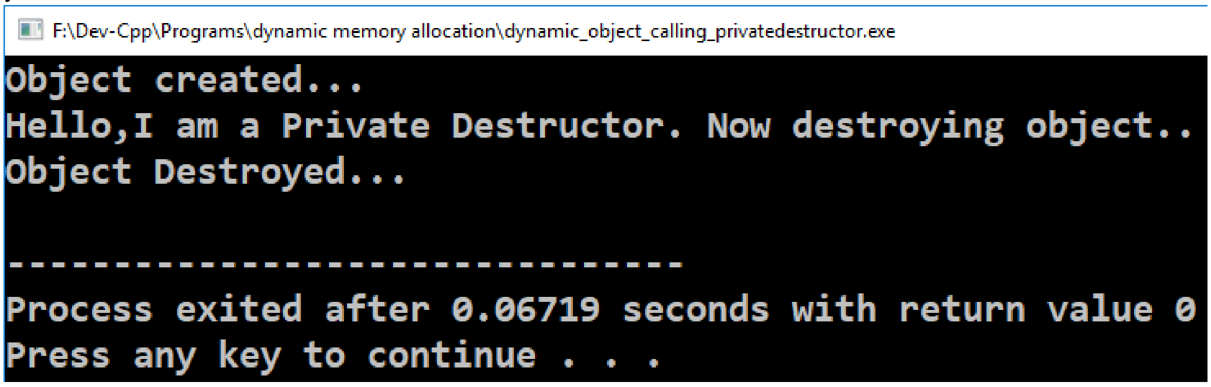


F:\Dev-Cpp\Programs\c++ programs\2019 session\dynamic memory\4.exe

```
new calls constructor to initilaise the object.
delete calls destructor to free the memory.

--------------------------------
Process exited after 0.05796 seconds with return value 0
Press any key to continue . . .
```

Program 8 to demonstrate that new operator can even call private destructor.

```
/*
WAP such that non-dynamic allocation of object should generate
compilation error. Idea is to create a private destructor in the
class. Non-dynamic objects will not be able to call private
destructors but dynamic objects can call.
When we make private destructor, the complier would generate error
for non-dynamically allocated objects because compiler need to
remove them from stack once they are not in use.
*/
#include<iostream>
using namespace std;
class Test
{
	public:
		Test()
		{
			cout<<"Object created...\n";
		}
	private:
		~Test()
		{
			cout<<"Hello,I am a Private Destructor. Now destroying object..\n";
		}
	friend void destructTest(Test*);
};
void destructTest(Test *ptr)
{
	delete ptr;
	cout<<"Object Destroyed...\n";
}
int main()
{
	//Test t1; //should generate compilation error
	Test *ptr =new Test;
	destructTest(ptr);
	return 0;
}
```



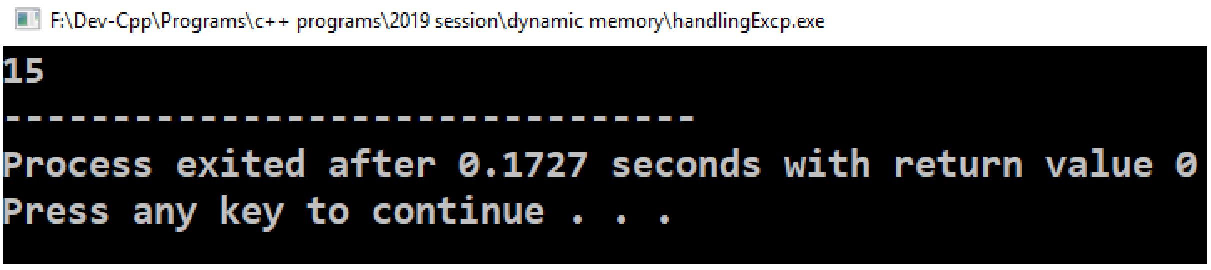
F:\Dev-Cpp\Programs\dynamic memory allocation\dynamic_object_calling_privatedestructor.exe

```
Object created...
Hello,I am a Private Destructor. Now destroying object..
Object Destroyed...


--------------------------------
Process exited after 0.06719 seconds with return value 0
Press any key to continue . . .
```

Exception Handling in Dynamic memory allocation

If there is insufficient available memory to fill an allocation request, then new will fail and a bad_alloc exception will be generated. This exception is defined in the header <new>. Your program should handle this exception and take appropriate action if a failure occurs. If this exception is not handled by your program, then your program will be terminated.

```
#include<iostream>
#include<new>
using namespace std;
int main()
{
	int *p=NULL;
	try
	{
		p= new int(15);
	}
	catch(bad_alloc ex)
	{
		cout<<"Allocation failure...";
		return 1;
	}
	cout<<*p;
	return 0;
}
```



```
F:\Dev-Cpp\Programs\c++ programs\2019 session\dynamic memory\handlingExcp.exe
15
--------------------------------
Process exited after 0.1727 seconds with return value 0
Press any key to continue . . .
```

My Notes: