|  |  |
| --- | --- |
| **Name** |  |
| **CMS ID** |  |
| **Date** |  |

In this lab, you will practice the following topics

1. Dynamic memory allocation
2. Constructors and destructors
3. **Dynamic memory allocation**

We all know the limitations of static variables. They are created during compilation. The main issue with them is their fixed size that can’t be changed. It might result in over use of memory if array size declared is too large compared to our input data whereas it might result in missing the representation of some data if our array declared have a small size. Dynamic memory allocation allows you to create variables during run-time process that fulfills your need. In order to create dynamic variable of some type, we need a pointer of that data type. The generalized syntax of creation of an int variable dynamically is

int \*ptrInt = new int;

In order to create a dynamic array of int, the syntax is

int \*ptr = new int[size];

Similarly, you can create dynamic variables and arrays of different datatypes.

The advantage of dynamic variables is that they can be deleted when they aren’t needed anymore. For example, if we want to delete the dynamic variable whose address is stored in ptrInt, we have to write

delete ptrInt;

And if we want to delete the dynamic array, we have to write

delete []ptr;

For further details, please see Lecture 2 and Lecture 3 that were delivered in class.

Task 1. Create a dynamic array of integer type and with the size of 3. Prompt the user to enter a number and after entering each number, ask user to enter Y to continue and any other key to exit. If the number of integer entered exceeds the size of array, create another dynamic array of doubled size, copy the content of previous array into new array, delete the previous array and keep getting data for the new one.

For example, initially the size of dynamic array will be 3, however, if user enters 4th number, you cant store it in array. In that case, create another array with size of 6, copy the three numbers stored in first array, delete the first array and keep getting data from user. In case if user enters 7th number, create another array of size 12 and continue the process.

1. **Constructors and destructors**

For the concepts of constructors, please see slides provided or read the following link

<https://www.studytonight.com/cpp/constructors-and-destructors-in-cpp.php>

Task 2: This task is about relationship between a book and library. A book usually have the following details

1. Name of book
2. Name of author
3. Name of publisher
4. ISSBN number
5. Edition

A library on the other hand have several books. We might have different libraries for different disciplines such as Mathematics, English literature, Computer Science, Electrical etc. Every library have a collection of the books. Now let’s discuss this problem using the concepts that we learnt so far.

You are required to create two classes named Book and Library. Class named book must have the following members and methods

**Book Class:**

**Private:**

String Name\_book, Name\_author, Name\_publisher, AssignedCMS; int ISSBN, Edition; bool isAvailable;

**Public:**

1. A default **constructor** with no parameters. It should assign true to isAvailable.
2. A **constructor** that takes four parameters named Name\_bookFun, Name\_authorFun, Name\_pubFun and ISSBNFun and assign them to the four private members. Absence of Edition number shows that this is the first edition of the book. It should also initialize isAvalable by true.
3. A **constructor** that takes five parameters named Name\_bookFun, Name\_authorFun, Name\_pubFun, ISSBNFun and EditionFun and assign them to the five private members. It should also initialize isAvalable by true.
4. A function named **inputDetails** that stores the values into the private members of the class.
5. Member function named **DispDetails** that takes a string named Name\_bookFun and display details of book if name is matched with the book name. It should also return true if the book is found and vice versa.
6. An overloaded function named **DispDetails** that takes an integer named ISSBNFun as input and perform the similar operation. It should also return true if the book is found and vice versa.
7. Another overloaded function named **DispDetails** that takes no arguments and display the details of the book.
8. A function named **AvailabilityStatus** that returns the value of **isAvailable**.
9. A function named **issueBook** that takes cms id of a student as input, store it in AssignedCMS, assign false to isAvailable. It should return an error message that “Book is already issued to a student with CMS id \_\_\_\_\_\_” if book is already assigned to someone else.

**Library Class:**

**Private:**

String Librarian; int maxBookNums, currBooksNum; Book \*booksPtr;

**Public:**

1. A **constructor** that takes a string named LibrarianFun as input, assigns it to Librarian. It should assign a default value of 100 to the variable named maxBookNums and create a dynamic array of size maxBookNums using booksPtr. It should also assign a value of 0 to currBooksNum.
2. A **constructor** that takes a string named LibrarianFun and an integer named maxSizeFun as inputs, assigns them to Librarian and maxBookNums. It should then create a dynamic array of size maxBookNums using booksPtr. It should also assign a value of 0 to currBooksNum.
3. Member function named **DispBookDetails** that display the details of all books in the library by calling **DispDetails** function with no arguments.
4. A function named **addBook** that takes all the details required for a book and add that book to the dynamic array of objects. It should also increment currBooksNum as a book is added. Make sure that currBooksNum doesn’t exceed the maximum size.
5. A function named **searchBook** that takes name of the book as input and display whether book is available or not.
6. A function named **assignBook** that takes cms id and book name as input and assign that book to the person with cms id if book is available. An error message should be displayed in case if book isn’t included in Library or is already assigned to someone else.
7. A destructor that destroy the dynamic array.