**Department of Computing**

**School of Electrical Engineering and Computer Science**

**CS250 – Data Structures and Algorithms**



**Lab 2: Dynamic Memory Allocation (I)**

**Submission Details**

|  |  |
| --- | --- |
| Name | CMS ID |
| Muhammad Umer | 345834 |
| Group | GP – 1 |
| Lab Engineer | Anum Asif |
| Faculty Member | Bostan Khan |
| Class | BEE12 |
| Date | 11/02/2024 |
| Time | 10:00 am – 12:50 pm |

**Table of Contents**

[2 Dynamic Memory Allocation (I) 3](#_Toc158632044)

[2.1 Introduction 3](#_Toc158632045)

[2.2 Objectives 3](#_Toc158632046)

[2.3 Tools/Software Requirement 3](#_Toc158632047)

[2.4 Description 3](#_Toc158632048)

[2.5 Deliverables 4](#_Toc158632049)

[3 Lab Tasks 5](#_Toc158632050)

[3.1 Task 1 5](#_Toc158632051)

[3.2 Task 2 6](#_Toc158632052)

[3.3 Task 3 7](#_Toc158632053)

[4 Conclusion 8](#_Toc158632054)

# Dynamic Memory Allocation (I)

## Introduction

This lab is about dynamic memory allocation. Dynamic memory allocation in C/C++ refers to performing memory allocation manually by programmer. Dynamically allocated memory is allocated on Heap and non-static and local variables get memory allocated on Stack.

## Objectives

This lab will revise the old concepts taught to the students in the previous semesters.

## Tools/Software Requirement

* Visual Studio C++

## Description

Consider two variants of declaring arrays below. Memory for the first variant gets allocated on the Stack. The lifetime of an array created using **Method A** depends on its scope. If it is defined globally, its life is equal to the lifetime of the application. If it is declared in a function, memory for it gets allocated on the stack when the function gets called. It gets deallocated when the function call terminates. All the data related to the function call, including the array gets removed from the stack. On the other hand, memory for the array created using the new operator gets allocated on the Heap at runtime. The lifetime of such an array is at max equal to the execution time of the application. If the array is no longer required, the memory allocated for it can be freed using delete[] command.

**Method A**

const int size=5;

int x[size];

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

{

//cout << "x[" << i << "] = ";

x[i] = i + 1;

}

**Method B**

int size; // Note that size variable is const in variant A whereas it isn’t in variant B. Find out the logic behind it.

cout << "Enter size of array: ";

cin >> size;

int \*x = new int[size];

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

{

//cout << "x[" << i << "] = ";

x[i] = i + 1;

}

## Deliverables

Compile a single word document by filling in the solution parts and submit this file on LMS. The name of word document should follow this format. i.e., YourFullName(reg)\_Lab#. You must show the implementation of the tasks in a complete manner to get your work graded.

***Note: Students are required to upload the lab on LMS before deadline.***

# Lab Tasks

## Task 1

Assume that the following variable declaration has already been made:

char\* oddOrEven = " Never odd or even ";

Write a single statement to accomplish each of the following tasks (assuming for each one that the previous ones have already been run). Make sure you understand what happens in each of them.

#include <iostream>

using *namespace* std;

*int* main(*void*)

{

*char* \*oddOrEven = " Never odd or even ";

    return 0;

}

1. Create a pointer to a char value named nthCharPtr pointing to the 6th character of oddOrEven (remember that the first item has index 0). Use the indexing operator.

*// 1*

*char* \*nthCharPtr = &oddOrEven[5];

1. Using pointer arithmetic, update nthCharPtr to point to the 4th character in oddOrEven.

*// 2*

nthCharPtr -= 2;

1. Print the value currently pointed to by nthCharPtr.

*// 3*

cout << \*nthCharPtr << endl; *// dereferencing the pointer*

root@Zonularity:/home/zonularity/dsa# cd "/home/zonularity/dsa/lab\_2/" && g++ task\_1.cpp -o task\_1 && "/home/zonularity/dsa/lab\_2/"task\_1

v

1. Create a new pointer to a pointer (a char \*\*) named pointerPtr that points to nthCharPtr.

*// 4*

*char* \*\*pointerPtr = &nthCharPtr;

## Task 2

Rewrite this program using pointers in place of arrays (use dynamic memory allocation operators new and delete). The syntax is int \*salArray= new int[size]; and for deletion delete [] salArray.

#include <iostream>

using namespace std;

int main(void)

{

int salary[20];

inti;

for (i = 0; i < 20; ++i)

{

cout << "Enter Salary: ";

cin >> salary[i];

}

for (i = 0; i < 20; ++i)

salary[i] = salary[i] + salary[i] / (i + 1);

return 0;

}

Code

#include <iostream>

using *namespace* std;

*int* main(*void*)

{

*int* \*salary = new *int*[20]; *// returns memory loc.*

*int* i;

    for (i = 0; i < 20; ++i)

    {

        cout << "Enter Salary: ";

        cin >> \*(salary + i); *// setting the value of the pointer*

    }

    for (i = 0; i < 20; ++i)

        \*(salary + i) = \*(salary + i) + \*(salary + i) / (i + 1);

    for (i = 0; i < 20; ++i)

        cout << "Salary: " << \*(salary + i) << endl;

    delete[] salary; *// free the memory*

    return 0;

}

Output

root@Zonularity:/home/zonularity/dsa# cd "/home/zonularity/dsa/lab\_2/" && g++ task\_2.cpp -o task\_2 && "/home/zonularity/dsa/lab\_2/"task\_2

Enter Salary: 130

Enter Salary: 240

Enter Salary: 450

Enter Salary: 230

Enter Salary: 450

Enter Salary: 230

Enter Salary: 450

Enter Salary: 230

Enter Salary: 450

Enter Salary: 560

Enter Salary: 670

Enter Salary: 340

Enter Salary: 560

Enter Salary: 340

Enter Salary: 560

Enter Salary: 230

Enter Salary: 560

Enter Salary: 340

Enter Salary: 230

Enter Salary: 560

Salary: 260

Salary: 360

Salary: 600

Salary: 287

Salary: 540

Salary: 268

Salary: 514

Salary: 258

Salary: 500

Salary: 616

Salary: 730

Salary: 368

Salary: 603

Salary: 364

Salary: 597

Salary: 244

Salary: 592

Salary: 358

Salary: 242

Salary: 588

## Task 3

Write a C++ program to get GPA of n number of students and display it, where n is the number of students entered by the user

Code

#include <iostream>

using *namespace* std;

*int* main(*void*)

{

*int* n, i;

    cout << "Enter the number of students: ";

    cin >> n;

*float* \*gpa = new *float*[n];

    for (i = 0; i < n; ++i)

    {

        cout << "Enter GPA of student " << i + 1 << ": ";

        cin >> \*(gpa + i); *// setting the value of the pointer*

*// ask for the GPA again if it is invalid*

        if (\*(gpa + i) < 0 || \*(gpa + i) > 4)

        {

            cout << "Invalid GPA. Please enter a valid GPA." << endl;

            --i;

        }

    }

*// display the GPA of the students*

    for (i = 0; i < n; ++i)

    {

        cout << "GPA of student " << i + 1 << ": " << \*(gpa + i) << endl;

    }

    delete[] gpa; *// free the memory*

    return 0;

}

Output

root@Zonularity:/home/zonularity/dsa/lab\_2# cd "/home/zonularity/dsa/lab\_2/" && g++ task\_3.cpp -o task\_3 && "/home/zonularity/dsa/lab\_2/"task\_3

Enter the number of students: 5

Enter GPA of student 1: 4.5

Invalid GPA. Please enter a valid GPA.

Enter GPA of student 1: 4.0

Enter GPA of student 2: 3.42

Enter GPA of student 3: 3.67

Enter GPA of student 4: 2.98

Enter GPA of student 5: 3.76

GPA of student 1: 4

GPA of student 2: 3.42

GPA of student 3: 3.67

GPA of student 4: 2.98

GPA of student 5: 3.76

# Conclusion

In this lab, we explored the concepts of dynamic memory allocation in C/C++, gaining hands-on experience managing memory usage beyond static and local variables. By allocating memory on the heap, we saw how programmers can adapt memory needs during runtime, offering flexibility and control over data structures. This exercise provided a foundation for understanding memory management in C/C++, a crucial skill for building efficient and adaptable programs.