**Department of Computing**

**School of Electrical Engineering and Computer Science**

**CS250 – Data Structures and Algorithms**



**Lab 3: Dynamic Memory Allocation (II)**

**Submission Details**

|  |  |
| --- | --- |
| Name | CMS ID |
| Muhammad Umer | 345834 |
| Group | GP – 1 |
| Lab Engineer | Anum Asif |
| Faculty Member | Bostan Khan |
| Class | BEE12 |
| Date | 19/02/2024 |
| Time | 10:00 am – 12:50 pm |

**Table of Contents**

[2 Dynamic Memory Allocation (II) 3](#_Toc159232500)

[2.1 Introduction 3](#_Toc159232501)

[2.2 Objectives 3](#_Toc159232502)

[2.3 Tools/Software Requirement 3](#_Toc159232503)

[2.4 Description 3](#_Toc159232504)

[2.5 Deliverables 4](#_Toc159232505)

[3 Lab Tasks 5](#_Toc159232506)

[3.1 Task 1 5](#_Toc159232507)

[3.2 Task 2 6](#_Toc159232508)

[3.3 Task 3 8](#_Toc159232509)

[4 Conclusion 9](#_Toc159232510)

# Dynamic Memory Allocation (II)

## 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

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 2

Complete the two parts for analyze pointer problem in the Lab\_3\_-\_Problem\_2.cpp file.

**Part 1:** Write a function void analyze\_pointer(int \*ptr) that does two things:

* Write the memory location pointed by the pointer to the console.
* Write the value of the integer (which the pointer points to) to the console.

**Part 2:** Use the function to complete two tasks:

* Allocate an int on the stack (e.g., “int iValue;"), assign a value to it, and get its memory location (with the reference operator—&) to pass this value to analyze\_pointer.
* Allocate an int on the heap (with the new operator). Assign a value to it, and pass it to analyze\_pointer.

Explain all the outputs (results of cout << in the main() method) and the reasons behind getting those outputs.

PART I Code

#include <iostream>

using *namespace* std;

*void* analyze\_pointer(*int* *\*ptr*)

{

    cout << "Memory location: " << *ptr* << endl;

    cout << "Value: " << \**ptr* << endl;

}

PART II Code

*int* main(*void*)

{

*int* iValue = 5;

*int* \*pValue = new *int*;

    \*pValue = 10;

    analyze\_pointer(&iValue);

    analyze\_pointer(pValue);

    delete pValue;

    return 0;

}

PART II Output

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

Memory location: 0x7ffd26a25e6c

Value: 5

Memory location: 0x559e41cd3eb0

Value: 10

Explanation

In the main method, we have two integer variables iValue and \*pValue. iValue is allocated on the stack and pValue is allocated on the heap. We pass the memory location of iValue and pValue to the analyze\_pointer function. The analyze\_pointer function prints the memory location and the value of the integer. The output of the program will be:

Memory location: 0x7ffd26a25e6c

Value: 5

Memory location: 0x559e41cd3eb0

Value: 10

The first output is the memory location of iValue and its value, and the second output is the memory location of pValue and its value. The memory location of iValue is on the stack, and the memory location of pValue is on the heap. The values of iValue and pValue are 5 and 10, respectively.

## Task 3

Define a struct Area that has two private variable members; units of type string and area\_value of type float. Modify the task\_3.cpp program to create a dynamic variable of type Area.

* Input from the keyboard the area\_value and its units. Compute one-half and one quarter of the area and display the results
* Destroy the dynamic variable at the end

Code

#include <iostream>

#include <string>

using *namespace* std;

*struct* Area

{

*private:*

    string units;

*float* area\_value;

*public:*

*void* set\_area\_value(*float* *value*)

    {

        area\_value = *value*;

    }

*void* set\_units(string *unit*)

    {

        units = *unit*;

    }

*float* get\_area\_value()

    {

        return area\_value;

    }

    string get\_units()

    {

        return units;

    }

};

*int* main(*void*)

{

    Area \*area = new Area;

*float* half\_area, quarter\_area, area\_value;

    cout << "Enter the area value: ";

    cin >> area\_value;

    area->set\_area\_value(area\_value);

    cout << "Enter the units: ";

    string units;

    cin >> units;

    area->set\_units(units);

    half\_area = area->get\_area\_value() / 2;

    quarter\_area = area->get\_area\_value() / 4;

    cout << "Half area: " << half\_area << " " << area->get\_units() << endl;

    cout << "Quarter area: " << quarter\_area << " "

         << area->get\_units() << endl;

    delete area;

    return 0;

}

Output

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

Enter the area value: 30

Enter the units: m^2

Half area: 15 m^2

Quarter area: 7.5 m^2

# 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.