CIS 200 - Lab0502

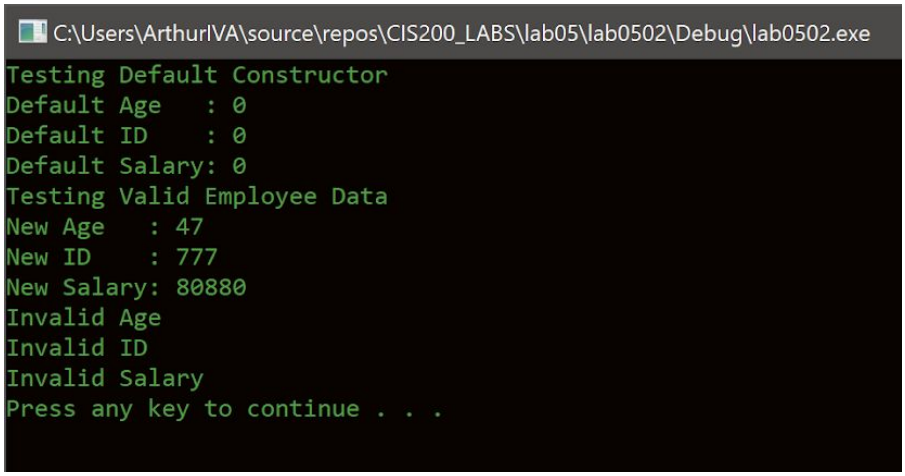
UML Diagram

employee
- age : integer - id : integer - salary : float
+ setAge(int input) : void + setID(int input) : void + setSalary(float input) : void + getAge() : integer + getID() : integer + getSalary() : float

Class Test Plan Version

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass/Fail
Valid Data	1	Default Constructor	Default	Age = 0 ID = 0 Salary = 0.0	Age = 0 ID = 0 Salary = 0.0	Pass
Valid Data	2	Set/Get Age	47	Age = 47	Age = 47	Pass
Valid Data	3	Set/Get ID	777	ID = 777	ID = 777	Pass
Valid Data	4	Set/Get Salary	80,880	Salary = 80880	Salary = 80880	Pass
Invalid Data	1	Set Age	-1	Invalid Age	Invalid Age	Pass
Invalid Data	2	Set ID	-777	Invalid ID	Invalid ID	Pass
Invalid Data	3	Set Salary	-100000	Invalid Salary	Invalid Salary	Pass

Screenshot



Class Algorithm

1. Create Class Employee
    - *a.* Create Function
        - i. Name: setAge
        - ii. Parameters: integer input
        - iii. Return: void
    - *b.* Create Function
        - i. Name: setID
        - ii. Parameters: integer input
        - iii. Return: void
    - *c.* Create Function
        - i. Name: setSalary
        - ii. Parameters: float input
        - iii. Return: void
    - *d.* Create Function
        - i. Name: getAge
        - ii. Parameters: n/a
        - iii. Return: integer
    - *e.* Create Function
        - i. Name: getID
        - ii. Parameters: n/a
        - iii. Return: integer
    - *f.* Create Function
        - i. Name: getSalary
        - ii. Parameters: n/a
        - iii. Return: float

2. Create Main
    a. Test Default Constructor
    b. Test Valid Employee Data
    c. Test Invalid Employee Data

Main Code For Above

_
[classTesting.cpp]

```
#include <iostream>
#include "employee.h"


int main()
{
	employee test;

	//Test Default Constructor
	std::cout << "Testing Default Constructor" << std::endl;
	std::cout << "Default Age   : " << test.getAge() << std::endl;
	std::cout << "Default ID    : " << test.getID() << std::endl;
	std::cout << "Default Salary: " << test.getSalary() << std::endl;

	//Test Valid Employee Fill
	test.setAge(47);
	test.setID(777);
	test.setSalary(80880);

	std::cout << "Testing Valid Employee Data" << std::endl;
	std::cout << "New Age   : " << test.getAge() << std::endl;
	std::cout << "New ID    : " << test.getID() << std::endl;
	std::cout << "New Salary: " << test.getSalary() << std::endl;

	//Test Invalid Employee Fill
	test.setAge(-1);
	test.setID(-777);
	test.setSalary(-100000);

	system("pause");
	return 0;
}
```

Main Test Plan (Provided Data)

Test Strategy	#	Description	Input	Expected Output	Actual Output	Pass /Fail
Valid Data	1	Data Provided	Table 1*	Employee Age ID Salary 1 30 111 30000 2 31 112 31000 3 32 113 32000 4 33 114 33000 5 34 115 34000 6 35 116 35000 Averages 32.5 32500	Employee Age ID Salary ---------- 1 30 111 30000 2 31 112 31000 3 32 113 32000 4 33 114 33000 5 34 115 34000 6 35 116 35000 ---------- Averages 32.5 32500	Pass

Screenshot

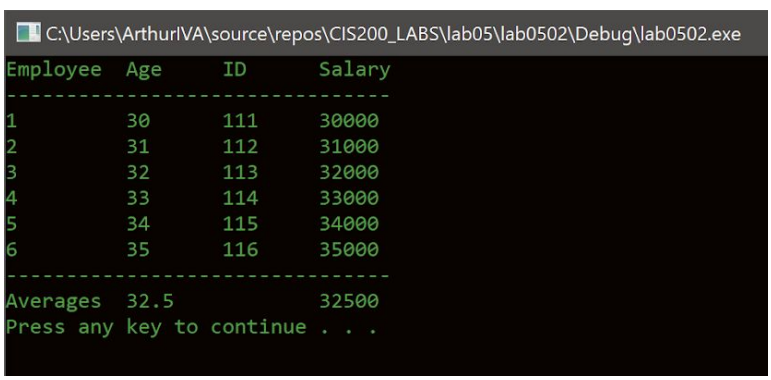


Table 1			
	Age	ID	Salary
x[0][0]	30	111	30000
x[0][1]	31	112	31000
x[0][2]	32	113	32000
x[1][0]	33	114	33000
x[1][1]	34	115	34000
x[1][2]	35	116	35000

All Other Code (Including Class and Header)

_
[employee.h]

```
#pragma once
#include <iostream>
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

class employee
{
private:
	int age;
	int id;
	float salary;

public:
	employee();		//Default Constructor

	//Setter Commands
	void setAge(int);
	void setID(int);
	void setSalary(float);

	//Getter Commands
	int getAge();
	int getID();
	float getSalary();

	~employee(); //Default Destructor
};
#endif EMPLOYEE_H
```

[employee.cpp]

```
#include "employee.h"

//Default Constructor
employee::employee()
{
	age = 0;
	id = 0;
	salary = 0.0;
}
```

```
//Setter Commands
void employee::setAge(int input)
{
	if (input > 0)
	{
		age = input;
	}
	else
	{
		std::cout << "Invalid Age" << std::endl;
	}
}

void employee::setID(int input)
{
	if (input > 0)
	{
		id = input;
	}
	else
	{
		std::cout << "Invalid ID" << std::endl;
	}
}

void employee::setSalary(float input)
{
	if (input >= 0)
	{
		salary = input;
	}
	else
	{
		std::cout << "Invalid Salary" << std::endl;
	}
}

//Getter Commands
int employee::getAge()
{
	return age;
}

int employee::getID()
{
	return id;
}
```

```
float employee::getSalary()
{
	return salary;
}

//Default Destructor
employee::~employee()
{
}
```

[arrayTesting.cpp]

```
#include <iomanip>
#include <iostream>
#include "employee.h"

const int MAX_ROWS = 2;
const int MAX_COLS = 3;

//Function Prototypes
void printEmployee(employee[MAX_ROWS][MAX_COLS], int, int);

int main()
{
	//Create 2 dim array of employees
	employee x[MAX_ROWS][MAX_COLS];

	//Set Ages
	x[0][0].setAge(30);
	x[0][1].setAge(31);
	x[0][2].setAge(32);
	x[1][0].setAge(33);
	x[1][1].setAge(34);
	x[1][2].setAge(35);

	//Set IDs
	x[0][0].setID(111);
	x[0][1].setID(112);
	x[0][2].setID(113);
	x[1][0].setID(114);
	x[1][1].setID(115);
	x[1][2].setID(116);

	//Set Salaries
	x[0][0].setSalary(30000);
	x[0][1].setSalary(31000);
	x[0][2].setSalary(32000);
```

```
	x[1][0].setSalary(33000);
	x[1][1].setSalary(34000);
	x[1][2].setSalary(35000);

	printEmployee(x, MAX_ROWS, MAX_COLS);

	system("pause");
	return 0;
}

//Description: Print out two dimensional array of employeee data.
//Pre-Condition: An array with employee age, ID, and salary
//Post-Condition: Printed!
void printEmployee(employee x[MAX_ROWS][MAX_COLS], int, int)
{
	//Counter
	int employCount = 0;

	//Totals
	int totalAge = 0;
	float totalSalary = 0;

	//Averages
	double averageAge = 0.0;
	float averageSalary = 0.0;


	//Table Header with columns
	std::cout << std::setw(10) << std::left << "Employee" << std::setw(8) << "Age"
<< std::setw(8) << "ID" << std::setw(8) << "Salary" << std::endl;
	std::cout << "--------------------------------" << std::endl;

	for  (int i = 0; i < MAX_ROWS; i++)
	{
		for (int j = 0; j < MAX_COLS; j++)
		{
			//Iterate Employee
			employCount++;
			//Print Formatted Data
			std::cout << std::setw(10) << std::left << employCount <<
std::setw(8) << x[i][j].getAge() << std::setw(8) << x[i][j].getID() << std::setw(8)
<< x[i][j].getSalary() << std::endl;
			//Compund Age
			totalAge += x[i][j].getAge();
			//Compund Salary
			totalSalary += x[i][j].getSalary();
		}
	}
```

```
	//Calculate Averages
	averageAge = totalAge / (double)employCount;
	averageSalary = totalSalary / (float)employCount;

	std::cout << "--------------------------------" << std::endl;
	std::cout << std::setw(10) << "Averages" << std::setw(8) << std::left << 
averageAge << std::setw(8) << "" << std::setw(8) << averageSalary << std::endl;

}
```

New Main Algorithm

1. Create Function
    i. Name: printEmployees
    ii. Parameters: Array of employee, rows, columns
    iii. Return: void
        1. Declare holder and counter variables
        2. Print out heading to table
        3. In nested FOR loop
            a. Print Data for the respective employee in order
            b. Iterate counter
            c. Compound Age
            d. Compound Salary
        4. Calculate Averages
        5. Print Averages
2. Create Main
    a. Declare 2 Dimensional Employee Array
    b. Set all Ages
    c. Set all IDs
    d. Set all Salaries
    e. printEmployees