# OOPS Lab File

**Name:** Prabhav Dogra

**Entry Number:** 19BCS058

**Course Code:** CSL 2022

**Submitted To:** Dr. Naveen Gondhi

# INDEX



**Compiler:** GCC Compiler TDM-GCC 4.9.2 64-bit release with commands "-g -std=c++11" while calling the compiler

**Google Drive Link with all assignment source files:**
https://drive.google.com/drive/folders/1i2RUg7n1_B-QprabjQPSCsY2VaFhukvv?usp=sharing

**1. Write a program to demonstrate the use of zero argument and parameterized constructors. [Constructors and Destructors]**

```
#include <bits/stdc++.h>
using namespace std;

class circle { // circle class
public :
	int radius ;
	circle() { // zero argument constructor
		cout << "Enter radius of the circle: " ;
		cin >> radius ;
	}
	circle( int r ) { // parameterised constructor
		radius = r ;
	}
	void print() {
		cout << "Radius: " << radius << '\n' ;
	}
};

int main()
{
	circle circle1(2), circle2 ;  // objects declaration
	circle1.print() ;
	circle2.print() ;
	return 0;
}
```

| **Input** | **Output** |
|---|---|
| 5 | Radius: 2<br>Radius: 5 |

**2. Write a program to demonstrate the use of dynamic constructor.**
**[Constructors and Destructors]**

```
#include <bits/stdc++.h>
using namespace std;

class item {
        int *i ;
        char *name ;
public:
        item (char *s) {      // dynamic constructor
                int length = strlen(s) ;
                name = new char[length] ;
                strcpy (name, s) ;
                i = new int ;
                cout << "Enter item number: " ;
                int x ;
                cin >> x ;
                *i = x ;
        }
        void print() {          // printing item number and name
                cout << "Item number: " << *i << " " << '\n' ;
                cout << "Item name: " << name << " " ;
                cout << '\n' ;
        }
};
int main() {
        char s[100];
        cout << "Enter item name: " ; // taking item name as input
        cin >> s ;
        item item1(s) ;
        item1.print() ;
        return 0 ;
}
```

| **Input** | **Output** |
|---|---|
| **Enter item name:** Cheese<br>**Enter item number:** 2 | **Item number:** 2<br>**Item name:** Cheese |

**3. Write a program to demonstrate the overloading of increment and decrement operators. [Operator Overloading]**

```
#include <bits/stdc++.h>
using namespace std;

class integer {
	int i ;
public:
	integer() { // zero argument constructor
		i = 0 ;
	}
	integer(int a) { // parameterized constructor
		i = a ;
	}
	integer& operator ++ () { // pre-increment
		i++ ;
		return *this ;
	}
	integer operator++(int dummy ) { // post-increment
		integer x = *this ;
		++(*this) ;
		return x ;
	}
	integer& operator -- () { // pre-decrement
		i-- ;
		return *this ;
	}
	integer operator -- (int dummy ) { // post-decrement
		integer x = *this ;
		--(*this) ;
		return x ;
	}
	void print() {
		cout << i << " "  ;
	}
};
```

```
int main() {
        integer i(3), j(3), k(3), l(3) ;
        cout << "Post-increment: " ; // printing the results
        cout << "i++: " ;
        (i++).print() ;
        cout << '\n' ;
        cout << "i: " ;
        (i).print() ;
        cout << '\n' << "Pre-increment: " ;
        cout << "++i: " ;
        (++j).print() ;
        cout << '\n' ;
        cout << "i: " ;
        j.print() ;
        cout << '\n' << "Post-decrement: " ;
        cout << "i--: " ;
        (k--).print() ;
        cout << '\n' ;
        cout << "i: " ;
        (k).print() ;
        cout << '\n' << "Pre-decrement: " ;
        cout << "--i: " ;
        (--l).print() ;
        cout << '\n' ;
        cout << "i: " ;
        l.print() ;
        return 0 ;
}
```

| **Input** | **Output** |
|---|---|
| - | Post-increment: i++: 3<br>i: 4<br>Pre-increment: ++i: 4<br>i: 4<br>Post-decrement: i--: 3<br>i: 2<br>Pre-decrement: --i: 2<br>i: 2 |

**4. Write a program to demonstrate the overloading of binary arithmetic operators. [Operator Overloading]**

```
#include <bits/stdc++.h>
using namespace std;

class point {
        float x, y;
public:
        point(float x = 0, float y = 0) { // constructor
                this ->x = x , this ->y = y ;
        }
        point operator + (point second) { // + operator overloading
                point ans ;
                ans.x = second.x + x ;
                ans.y = second.y + y ;
                return ans ;
        }
        void print() { // printing method inside the classs
                cout << "x: " << x << " " << "y: " << y << '\n' ;
        }
};
int main() {
        point p1(2.0,3.0), p2(5.0, 6.0) ;  // declaring objects
        cout << "Point 1: ";
        p1.print() ;                        // printing object p1
        cout << "Point 2: " ;
        p2.print() ;                        // printing object p2
        point p3 = p2 + p1 ;                // copy constructor
        cout << "Point 3: Point 1 + Point 2: " ;
        p3.print() ;                        // printing object p3
        return 0 ;
}
```

| **Input** | **Output** |
|---|---|
| **-** | **Point 1: x:** 2 **y:** 3<br>**Point 2: x:** 5 **y:** 6<br>**Point 3: Point 1 + Point 2: x:** 5 **y:** 6 |

**5. Write a program to demonstrate the overloading of memory management operators. [Operator Overloading]**

```
#include<bits/stdc++.h>
#include <iostream>
using namespace std;

class student
{
   string name;
   int entry_no;
public:
   student() { }  // zero argument constructor
   student(string n, int a) {
      name = n;
      entry_no = a;
   }
   void * operator new(size_t size) { // overloading new operator
      void * p = ::new student();
      return p;
   }

   void operator delete(void * p) {  // overloading delete operator
      free(p);
   }
   void display() {
      cout<< "Name:" << name << '\n' << "Entry Number:" << entry_no << '\n' ;
   }
};
int main()
{
   student * p = new student("Prabhav", 58);
   p->display();
   delete p;
}
```

| Input | Output |
|---|---|
| - | **Name:** Prabhav<br>**Entry Number:** 58 |