NUST CHIP DESIGN CENTRE

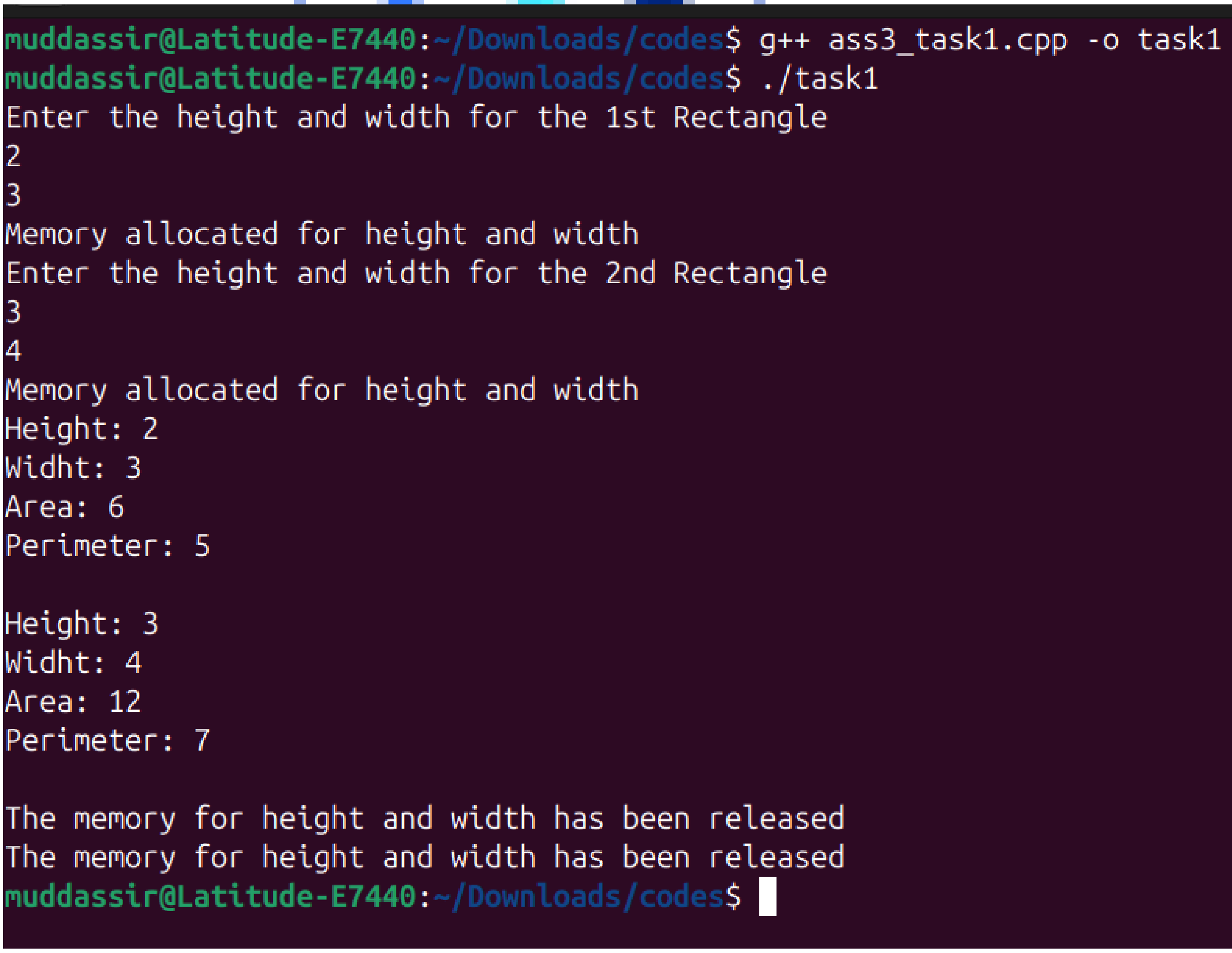
C / C++ Programming

Assignment # 3
Constructor in C++ & Catching Bugs in C++

Name	*Muddassir Ali Siddiqui*
Instructor	*Hira Sohail*
Date	*24th July 2025*

1. **In-Lab Tasks:** (*Write your lab task & screenshots here*)

   i. **Task 1:**

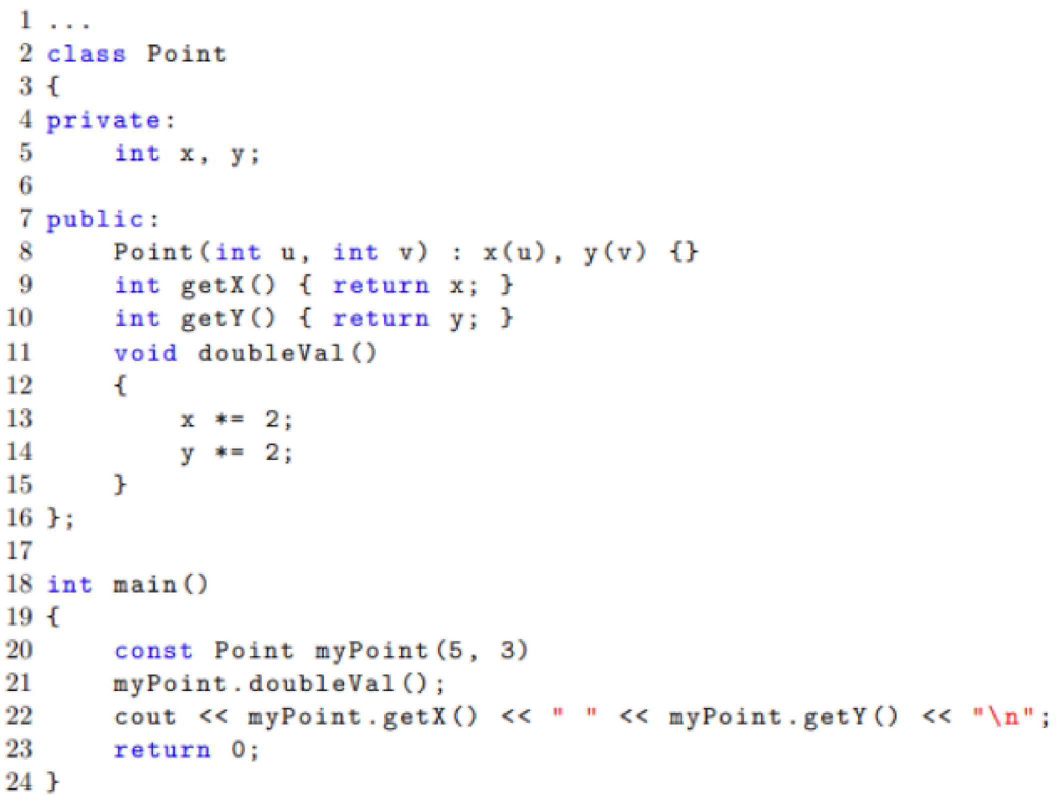


```
muddassir@Latitude-E7440:~/Downloads/codes$ g++ ass3_task1.cpp -o task1
muddassir@Latitude-E7440:~/Downloads/codes$ ./task1
Enter the height and width for the 1st Rectangle
2
3
Memory allocated for height and width
Enter the height and width for the 2nd Rectangle
3
4
Memory allocated for height and width
Height: 2
Widht: 3
Area: 6
Perimeter: 5

Height: 3
Widht: 4
Area: 12
Perimeter: 7

The memory for height and width has been released
The memory for height and width has been released
muddassir@Latitude-E7440:~/Downloads/codes$ 
```

   ii. **Task 2:**

`const Point myPoint (5, 3);` A `**const**` object is created, but it later tries to call `doubleVal()`, which is a non-const function that modifies object state — this is not allowed.

1.



```cpp
...
class Point
{
private:
    int x, y;

public:
    Point(int u, int v) : x(u), y(v) {}
    int getX() { return x; }
    int getY() { return y; }
    void doubleVal()
    {
        x *= 2;
        y *= 2;
    }
};

int main()
{
    const Point myPoint(5, 3);
    myPoint.doubleVal();
    cout << myPoint.getX() << " " << myPoint.getY() << "\n";
    return 0;
}
```

`void setX(int newX) const { x = newX; }`The `setX` function is marked as `const`, meaning it shouldn't modify any members, but it tries to modify `x`, which leads to a compiler error.

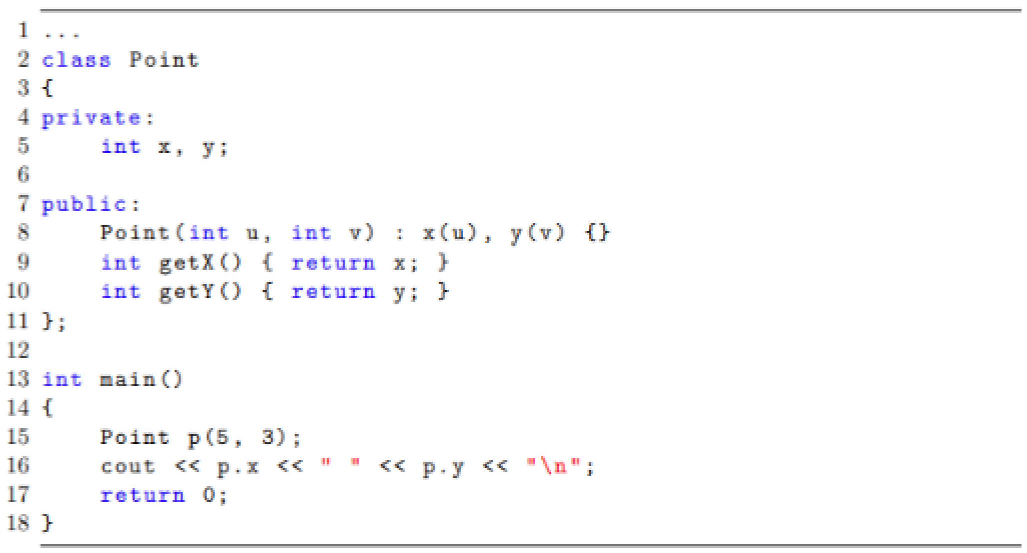
2.

```
...
class Point
{
private:
    int x, y;

public:
    Point(int u, int v) : x(u), y(v) {}
    int getX() { return x; }
    int getY() { return y; }
```

```
    void setX(int newX) const { x = newX; }
};

int main()
{
    Point p(5, 3);
    p.setX(9001);
        cout << p.getX() << ' ' << p.getY();
    return 0;
}
```

`cout << p.x << " " << p.y << "\n";`The program tries to access private members `x` and `y` directly, which violates encapsulation and leads to an access error.

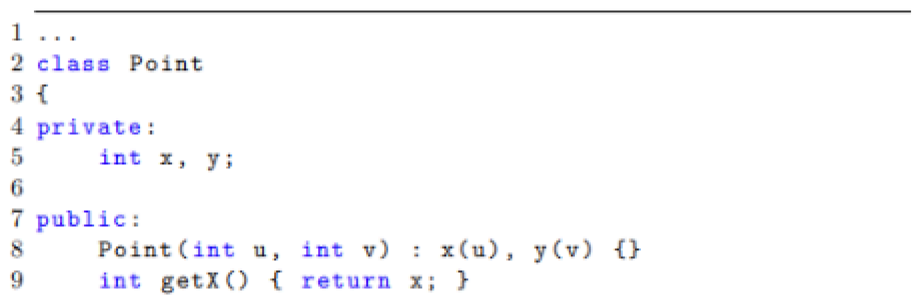
3.



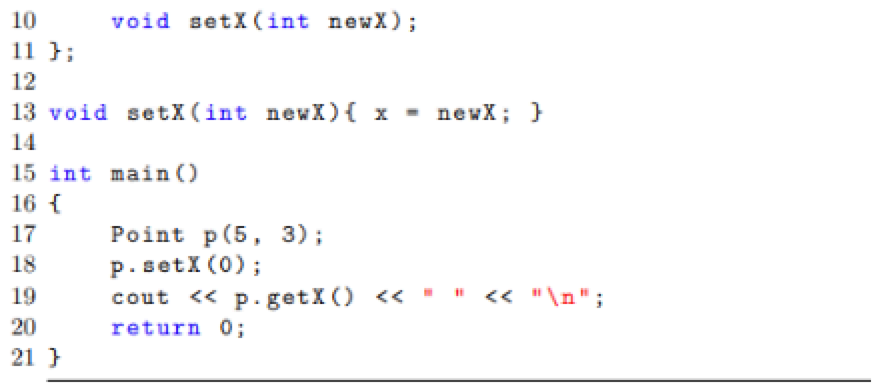
```
...
class Point
{
private:
    int x, y;

public:
    Point(int u, int v) : x(u), y(v) {}
    int getX() { return x; }
    int getY() { return y; }
};

int main()
{
    Point p(5, 3);
    cout << p.x << " " << p.y << "\n";
    return 0;
}
```

`void setX(int newX) { x = newX; }` This method is defined outside the class but is missing the `Point::` scope resolution operator, so the compiler doesn't know which class `x` belongs to.

4.



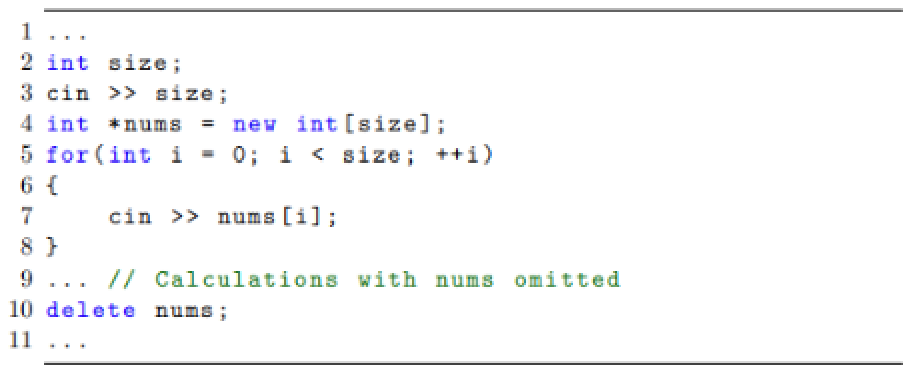
```
...
class Point
{
private:
    int x, y;

public:
    Point(int u, int v) : x(u), y(v) {}
    int getX() { return x; }
```



```
    void setX(int newX);
};

void setX(int newX){ x = newX; }

int main()
{
    Point p(5, 3);
    p.setX(0);
    cout << p.getX() << " " << "\n";
    return 0;
}
```

`delete nums;` The code uses `delete` instead of `delete[]` to deallocate a dynamically allocated array, which results in undefined behavior.

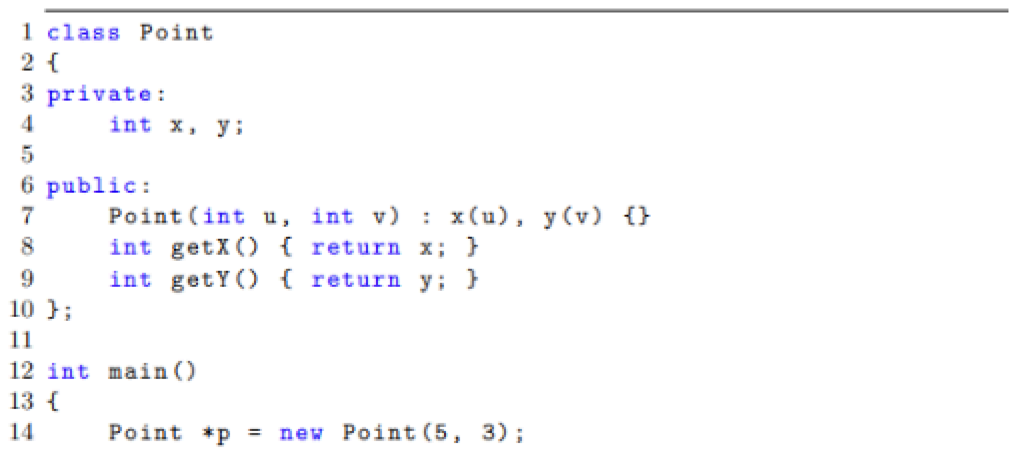
5.



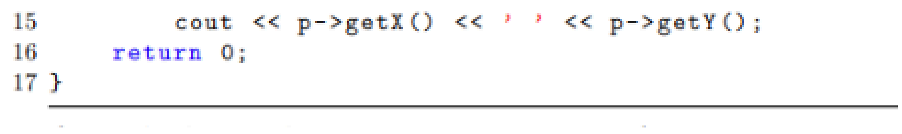
```
...
int size;
cin >> size;
int *nums = new int[size];
for(int i = 0; i < size; ++i)
{
    cin >> nums[i];
}
... // Calculations with nums omitted
delete nums;
...
```

`cout << p->getX() << << p->getY();`The output statement contains a syntax error due to the extra `<<`, making it invalid and uncompileable.

6.



```
class Point
{
private:
    int x, y;

public:
    Point(int u, int v) : x(u), y(v) {}
    int getX() { return x; }
    int getY() { return y; }
};

int main()
{
    Point *p = new Point(5, 3);
```



```
        cout << p->getX() << ' ' << p->getY();
    return 0;
}
```