## Assignment\_12-08-2024

Author: ThanhTH10

Date: 13/08/2024

1. pre increment vs post increment. Which one is the faster one? Overload pre increment and post increment. Explain the difference with respect to assembly code by using objdump command

| **Characteristic** | **Pre-increment (++x)** | **Post-increment (x++)** |
| --- | --- | --- |
| Value of the expression | Incremented value of the variable | Original value of the variable |
| Order of operations | Variable is incremented before the expression is evaluated | Variable is incremented after the expression is evaluated |
| Return value | Reference to the incremented variable | Original value of the variable |
| Assembly code | incl (%rax) | mov (%rax),%eax, incl (%rax) |
| Performance | Slightly faster | Slightly slower |
| Use case | Preferred when the incremented value is needed immediately | Preferred when the original value needs to be used first |

As for which one is faster, it largely depends on the specific use case and the underlying hardware/architecture. Generally, pre-increment is considered slightly faster because it doesn't require storing the original value before incrementing it.

To demonstrate the difference in assembly code, we can use the objdump command, which disassembles the compiled machine code.

#include <stdio.h>

int main() {

    int x = 0;

    printf("Pre-increment: %d\n", ++x);

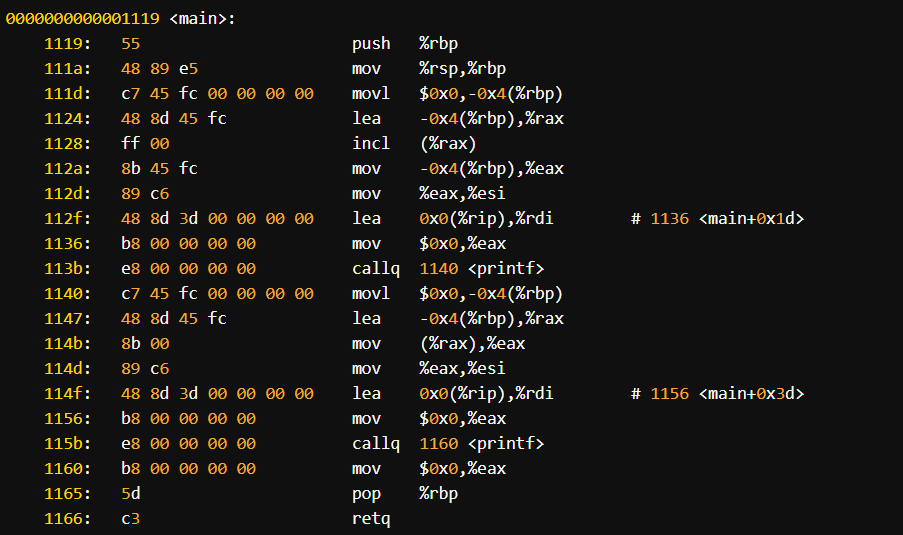
    x = 0;

    printf("Post-increment: %d\n", x++);

    return 0;

}

*Compiling the code and disassembling it using objdump -d gives us the following:*



In the pre-increment case (++x), we can see that the value is incremented before it is used (incl (%rax)), whereas in the post-increment case (x++), the original value is first used (mov (%rax),%eax), and then the variable is incremented (incl (%rax)).

Regarding overloading pre-increment and post-increment, we can do so by defining the corresponding operator overload functions in your class. The pre-increment operator overload function would have the signature T& operator++(), and the post-increment operator overload function would have the signature T operator++(int) (the int parameter is just a dummy parameter to differentiate it from the pre-increment overload).

class MyClass

{

    int value\_ = 0;

public:

    MyClass &operator++()

    {

        // Implement pre-increment logic here

        ++value\_;

        return \*this;

    }

    MyClass operator++(int)

    {

        // Implement post-increment logic here

        MyClass temp = \*this;

        ++value\_;

        return temp;

    }

};

The main difference between the two is that the pre-increment operator overload returns a reference to the modified object, while the post-increment operator overload returns a copy of the original object before the increment operation.

2. Implement hierarchical inheritance for the employee base class with derived classes it team, sales team, marketing team

#include <iostream>

using namespace std;

class Employee

{

protected:

    string name;

    int age;

    double salary;

public:

    Employee(string \_name, int \_age, double \_salary) : name(\_name), age(\_age), salary(\_salary) {}

    void display()

    {

        cout << "Name: " << name << "\t| Age: " << age << "\t| Salary: " << salary;

    }

};

class ITTeam : public Employee

{

    string it\_level;

public:

    ITTeam(string \_name, int \_age, double \_salary, string \_it\_level) : Employee(\_name, \_age, \_salary), it\_level(\_it\_level) {}

    void display()

    {

        Employee::display();

        cout << "\t| Level: " << it\_level << endl;

    }

};

class SalesTeam : public Employee

{

    double salesTarget;

public:

    SalesTeam(string \_name, int \_age, double \_salary, double \_salesTarget) : Employee(\_name, \_age, \_salary), salesTarget(\_salesTarget) {}

    void display()

    {

        Employee::display();

        cout << "\t| Sales Target: " << salesTarget << endl;

    }

};

class MarketingTeam : public Employee

{

    string marketingStrategy;

public:

    MarketingTeam(string \_name, int \_age, double \_salary, string \_marketingStrategy) : Employee(\_name, \_age, \_salary), marketingStrategy(\_marketingStrategy) {}

    void display()

    {

        Employee::display();

        cout << "\t| Maketing Stratery: " << marketingStrategy << endl;

    }

};

int main()

{

    ITTeam it\_team("Thanh", 22, 5000, "Junior");

    it\_team.display();

    SalesTeam sales\_team("Ngan", 22, 3600, 10000);

    sales\_team.display();

    MarketingTeam maket\_team("Ha", 23, 4000, "tech");

    maket\_team.display();

    return 0;

}

Output:

