Principles of Programming Language (CS302)

Assignment - 4

U19CS012

1.) Write a program in C++ that **Calls** both

- ✓ **Dynamically Bound** method
- ✓ **Statically Bound** method

Large number of times, timing the calls to both of the two. Compare the **Timing results** and Compute the difference of the time required by the Two. Explain the Results.

Binding - Mapping of one thing to another

[In Context of **Compiled Languages** - Link between a **Function Call** and **Function Definition**.] When a function is called in C++, the Program Control binds to the Memory Address where that Function is defined.

	Static Binding	Dynamic Binding
Happens at	**Compile** Time	**Run**time
Happens when	all information needed to call a function is available at the compile-time.	Happens when the compiler cannot determine all information needed for a function call at compile-time.
Achieved Using	Normal Function Calls, Function & Operator Overloading	Virtual functions
Execution Time	**Faster** {Since all info is Available before runtime}	**Slower** { Function call is not resolved until runtime for later binding}
Code Size	**Large**	**Small & Flexible** Code { single function can handle different types of objects at runtime}

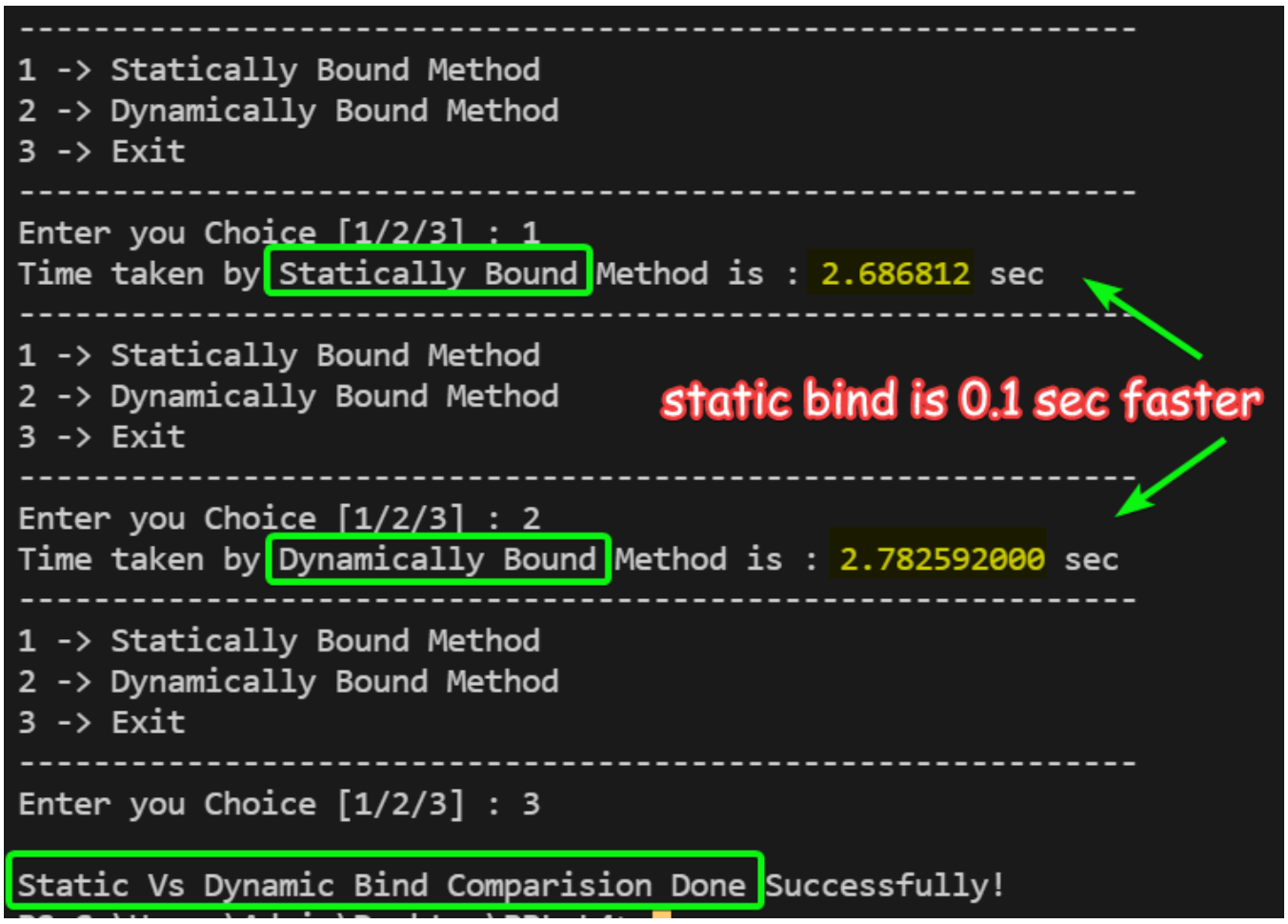
Code

```cpp
// For Basic IO
#include <iostream>
// For setprecision
#include <iomanip>
// For Time Calculation
#include <chrono>

using namespace std;
using namespace std::chrono;

// [U19CS012] BHAGYA VINOD RANA

// Maximum Number of Calls
const int MAX_CALLS = 1000000;

// Complex Calculation
void complex_calc()
{
    long long int sum = 0;
    for (int i = 0; i < 1000; i++)
        sum += i;
}

// Base Class
class base
{
public:
    // Statically Bound
    void fun_1() { complex_calc(); }
    // Dynamically Bound
    virtual void fun_2() { complex_calc(); }
};

// Derived Class
class derived : public base
{
public:
    // This was Just to Confuse
    void fun_1() { cout << "derived-1\n"; }
    void fun_2() { complex_calc(); }
};

void menu()
{
    cout << "------------------------------------------------------------\n";
    cout << "1 -> Statically Bound Method\n";
    cout << "2 -> Dynamically Bound Method\n";
    cout << "3 -> Exit\n";
```

```
    cout << "------------------------------------------------------------\n";
}

int main()
{
    base *p;
    base t;
    derived obj1;
    p = &obj1;

    auto start = high_resolution_clock::now();
    auto end = high_resolution_clock::now();
    double time_taken = duration_cast<nanoseconds>(end - start).count();

    int choice = 1;
    while (true)
    {
        menu();
        cout << "Enter you Choice [1/2/3] : ";
        cin >> choice;
        switch (choice)
        {
        case 1:
            // Static Bind
            start = high_resolution_clock::now();

            for (int i = 0; i < MAX_CALLS; i++)
                p->fun_1();

            end = high_resolution_clock::now();

            // Calculating total time taken by the Static Bind.
            time_taken = duration_cast<nanoseconds>(end - start).count();
            time_taken *= 1e-9;
            cout << "Time taken by Statically Bound Method is : " << fixed << time_taken <<
setprecision(9);
            cout << " sec" << endl;

            break;

        case 2:
            // Dynamic Bind
            start = high_resolution_clock::now();

            for (int i = 0; i < MAX_CALLS; i++)
            {
                p->fun_2();
            }

            end = high_resolution_clock::now();
```

```
            // Calculating total time taken by the Static Bind.
            time_taken = duration_cast<nanoseconds>(end - start).count();
            time_taken *= 1e-9;
            cout << "Time taken by Dynamically Bound Method is : " << fixed << time_taken <<
setprecision(9);
            cout << " sec" << endl;
            break;
        case 3:
            cout << "\nStatic Vs Dynamic Bind Comparision Done Successfully!\n";
            return 0;
            break;
        default:
            cout << "Enter Valid Input! Please Try Again!\n";
        }
    }
    return 0;
}
```

Output



We Observer that {Time Taken by **Dynamic Bind** takes More (>) time than **Static Bind**, Since in Dynamic Bind, the Function Call is Resolved at the **Run Time**}.

2.) Design and implement a C++ program that defines a base class A, which has a subclass B, which itself has a subclass C. The A class must implement a method, which is overridden in both B and C.

You must also write a test class that instantiates A, B, and C and includes three calls to the method.

- ✓ One of the calls must be **statically bound to A's** method.
- ✓ One call must be **dynamically bound to B's** method
- ✓ One must be **dynamically bound to C's** method.

All of the method calls must be through a pointer to class A.

Code

```
#include <iostream>
using namespace std;

class A
{
public:
    virtual void method()
    {
        cout << "Method From -> class A\n";
    }

    // Constructor
    A()
    {
        method();
    }
};

// B class is derived from base class A.
class B : public A
{
public:
    void method()
    {
        cout << "Method From -> class B\n";
    }
};

// C class is derived from base class B.
class C : public B
{
public:
    void method()
```

```
    {
        cout << "Method From -> class C\n";
    }
};

int main()
{
    A *a;

    cout << "\nCall that is Statically Bound to A's Method\n";
    A tmp;
    a = &tmp;

    cout << "\nCall Dynamically Bound to B's Method\n";
    B b;
    a = &b;
    a->method();

    cout << "\nCall Dynamically Bound to C's Method\n";
    C c;
    a = &c;
    a->method();

    return 0;
}
```

Output

```
Call that is Statically Bound to A's Method
Method From -> class A

Call Dynamically Bound to B's Method
Method From -> class A
Method From -> class B

Call Dynamically Bound to C's Method
Method From -> class A
Method From -> class C
```

3.) Consider the following C++ skeletal program [Question in Assignment had Errors]:

```
class Big
{
    int i;
    float f;
    void fun1() throw float
    {
        ...
        try
        {
            ...
            throw i;
            ...
            throw f;
            ...
        }
        // Correction in Asssignment {instead of 'float' -> 'int'}
        catch (int){ ... }
        ...
    }
};
```

Block 1

```
class Small
{
    int j;
    float g;
    void fun2() throw float
    {
        ...
        try
        {
            ...
            try
            {
                Big.fun1();
                ...
                throw j;
                ...
                throw g;
                ...
            }
            catch (int) { ... }
            ...
        }
        catch (float){ ... }
    }
};
```

Block 2

Block 3

In each of the four throw statements, where is the **exception handled**?

[**Note**: fun1() is called from fun2() in class **Small**.]

Throw Statement	Catch Block [1/2/3]	Reason
throw i;	1	Since 'i' is an **int** variable. The nearest matching - 'int' catch block after throw i catches the error. So, **Block 1** handles **throw** i.
throw f;	3	Since 'f' is **float** variable. The nearest matching – 'float' catch block after throw 'f' will catch the error because there is **no matching catch** available in fun1() and also fun1() is called by fun2(). So, it will catch by it. So, **Block 3** handles **throw** f.
throw j;	2	'j' is an **int** variable. The nearest matching - int catch block after throw j catches the error. So, **Block 2** handles **throw** j.
throw g;	3	Since 'g' is **float** variable. The nearest matching - float catch block after throw g will catch the error. So, **Block 3** handles throw g.

4.) Write a C++ program that takes a set of inputs.

The type of input governs the kind of operation to be performed, i.e. **concatenation for strings and addition for int or float**.

You need to write the class template **AddElements** which has a functions:

- **add()** for giving the sum of int or float elements.
- **concatenate()** to concatenate the second string to the first string.

Code

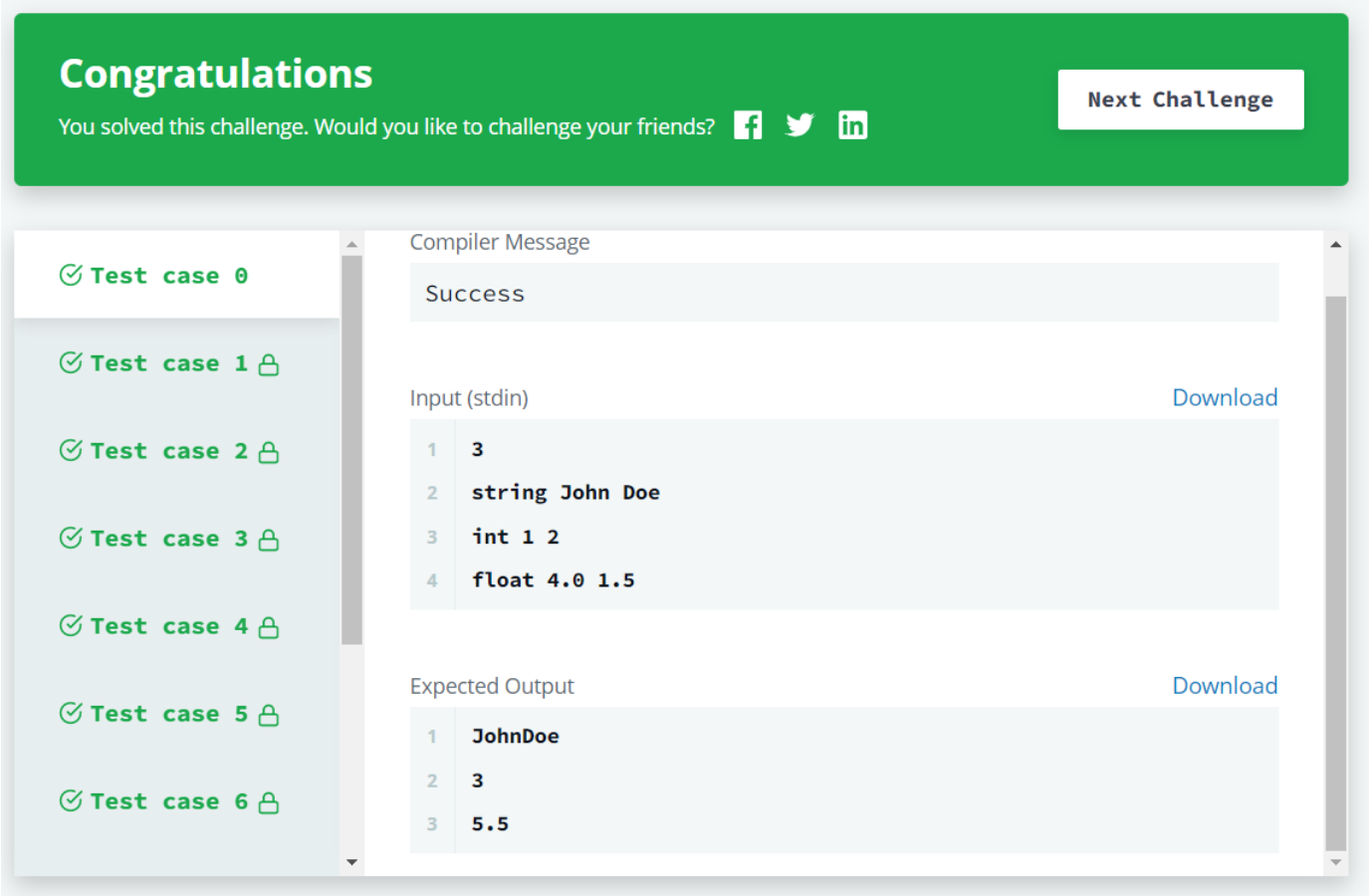
```cpp
#include <bits/stdc++.h>
using namespace std;
// [U19CS012] BHAGYA VINOD RANA

struct fast
{
    fast()
    {
        ios_base::sync_with_stdio(false);
        cin.tie(NULL);
    }
};

fast f;

/*Write the class AddElements here*/
template <class T>
class AddElements
{
private:
    T a;

public:
    // Constructor
    AddElements(T val)
    {
        a = val;
    }

    // Add for int and float
    T add(T &n)
    {
        return a + n;
    }
```

```
    // String Concatanation
    T concatenate(T b)
    {
        return a + b;
    }
};

int main()
{
    int n, i;
    cin >> n;
    for (i = 0; i < n; i++)
    {
        string type;
        cin >> type;
        if (type == "float")
        {
            double element1, element2;
            cin >> element1 >> element2;
            AddElements<double> myfloat(element1);
            cout << myfloat.add(element2) << endl;
        }
        else if (type == "int")
        {
            int element1, element2;
            cin >> element1 >> element2;
            AddElements<int> myint(element1);
            cout << myint.add(element2) << endl;
        }
        else if (type == "string")
        {
            string element1, element2;
            cin >> element1 >> element2;
            AddElements<string> mystring(element1);
            cout << mystring.concatenate(element2) << endl;
        }
    }
    return 0;
}
```

Output

```
3
string Bhagya Rana
BhagyaRana
int 10 2
12
float 2.5 5.5
8
```

Also Tested the Code on **Hacker-Rank** Website.



[Problem Link - https://www.hackerrank.com/challenges/c-class-templates/problem]

SUBMITTED BY: U19CS012

BHAGYA VINOD RANA