Exception Handling

In this week

- C++ switch-case statement
- C++ functions: Pre and Post conditions
- Exception Handling (Runtime Error Handling)

C++ switch-case statement

- Consider the following simple program with some conditional statements

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
    int x;
    srand(time(0));
    x = rand() % 5;
    if (x == 0)
        cout << "Value of x is 0" << endl;
    else if (x == 1)
        cout << "Value of x is 1" << endl;
    else if (x == 2)
        cout << "Value of x is 2" << endl;
    else if (x == 3)
        cout << "Value of x is 3" << endl;
    else
        cout << "Value of x is 4" << endl;
    system("Pause");
    return 0;
}
```

C++ switch-case statement

- C++ provides the **switch-case** statement to achieve the same result

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
    int x;
    srand(time(0));
    x = rand() % 5;
    switch (x)
    {
        case 0:
            cout << "Value of x is 0" << endl;
            break;
        case 1:
            cout << "Value of x is 1" << endl;
            break;
        case 2:
            cout << "Value of x is 2" << endl;
            break;
        case 3:
            cout << "Value of x is 3" << endl;
            break;
        default:
            cout << "Value of x is 4" << endl;
    }
    system("Pause");
    return 0;
}
```

C++ switch-case statement

- The swicth-case statement checks each of the cases listed and executes the case that matches the value of the switch statement
- An important aspect of the switch-case statement is that once a matching case is found, then the matching case **and all the cases below it** will be executed
- In order to only execute the matching case, a break statement is used
- The break statement forces execution to jump to the statement below the switch-case statement

C++ switch-case statement

- Now consider the following program and convert the if else-if else conditional statements to an equivalent switch-case statement

```
#include <iostream>
#include <ctime>
using namespace std;
int main()
{
    int x;
    srand(time(0));
    x = rand() % 11;
    if (x == 0 || x == 3 || x == 6 || x == 9)
        cout << x << " % 3 is 0" << endl;
    else if (x == 1 || x == 4 || x == 7 || x == 10)
        cout << x << " % 3 is 1" << endl;
    else
        cout << x << " % 3 is 2" << endl;
    system("Pause");
    return 0;
}
```

C++ switch-case statement

- All we need to realize is the fact that once a matching case is found, all the cases under it will be executed or until a break statement is found. Therefore the following code provides the required equivalent switch-case.

```
int main()
{
    int x;
    srand(time(0));
    x = rand() % 11;
    switch (x)
    {
        case 0:
        case 3:
        case 6:
        case 9:
            cout << x << " % 3 is 0" << endl;
            break;
        case 1:
        case 4:
        case 7:
        case 10:
            cout << x << " % 3 is 1" << endl;
            break;
        default:
            cout << x << " % 3 is 2" << endl;
    }
    system("Pause");
    return 0;
}
```

C++ functions
Pre-condition and Post-condition

- Generally speaking it is assumed that the parameters of a function will always receive correct data values from their corresponding arguments
- This is why we need to check the correctness of any value before using it as argument when calling a function
- That is a C++ function assumes its parameter will not cause any runtime error or errors in the function body
- This is usually emphasized by explicitly writing comments inside the function block that specify what conditions the parameters of the function must satisfy (known as the pre-condition) and what the function is intended to perform (known as the post-condition)
- Whenever a function is given arguments that violate its pre-condition requirements, then the function may get into a runtime error and crash the program or may give some wrong result
- The next example shows ***isPrime*** function stating explicitly its pre-condition and post-condition for clarity purposes

C++ functions
Pre-condition and Post-condition

```
bool isPrime(const int x)
{
    //Pre-condition:- x is an integer value greater than 1
    //Post-condition:- returns true if x is prime and false otherwise
    for (int k = 2; k < x; k++)
        if (x % k == 0)
            return false;
    return true;
}
int main()
{
    int n;
    cout << "Please enter a positive integer greater than 1: ";
    cin >> n;
    if (n <= 1)
        cout << "Please enter a correct integer." << endl;
    else
    {
        bool ans = isPrime(n);
        if (ans == true)
            cout << "The number is prime." << endl;
        else
            cout << "The number is not prime." << endl;
    }
    system("Pause");
    return 0;
}
```

Exception Handling

- But what if the parameters of a function violate the pre-condition?
- Obviously a runtime error may happen or may result to some semantic errors
- Sometimes, we may also opt to check the correctness of the parameters of a function and crash the program if the parameter doesn't satisfy the pre-condition
- One way to crash the program will be to use the **abort** function as shown below

Exception Handling

```
bool isPrime(const int x)
{
    //Pre-condition: x is an integer value greater than 1
    //Post-condition: returns true if x is prime and false otherwise
    if (x < 2)
    {
        cout << "An integer greater than 1 expected." << endl;
        abort();
    }
    else
    {
        for (int k = 2; k < x; k++)
            if (x % k == 0)
                return false;
        return true;
    }
}
```

Exception Handling

- Another way to test the parameter is to use the **assert** built-in function
- This function requires an include directive

  #include <cassert>
- The assert function takes one argument which is often a Boolean expression testing some condition
- If the Boolean expression is evaluated to true then the assert function does nothing and execution continues to the next statement
- If the Boolean expression is evaluated to false then the assert function prints a description of the error message and then aborts the program. See below

Exception Handling

- Observe that we don't really need to print our own error message because the assert function will print a nice error message

```
bool isPrime(const int x)
{
    //Pre-condition: x is an integer value greater than 1
    //Post-condition: returns true if x is prime and false otherwise
    assert(x > 1);
    for (int k = 2; k < x; k++)
        if (x % k == 0)
            return false;
    return true;
}
```

Exception Handling

- Yet another way to check the correctness of any variable (including a parameter) in a program is to use **try-catch blocks**
- In the try-catch block, the code that we need to execute is placed in a try block
- If any variable is not correct, we throw an exception and the code in the try block is automatically skipped and execution goes to the catch block which will handle the error
- See below

Exception Handling

```
bool isPrime(const int x)
{
    //Pre-condition: x is an integer value greater than 1
    //Post-condition: returns true if x is prime and false otherwise
    if (x < 2)
        throw(x);
    for (int k = 2; k < x; k++)
        if (x % k == 0)
            return false;
    return true;
}
int main()
{
    int x;
    srand(time(0));
    x = rand() % 11 - 5;
    try
    {
        if (isPrime(x) == true)
            cout << x  << " is prime." << endl;
        else
            cout << x << " is not prime." << endl;
    }
    catch (int e)
    {
        cout << "An integer greater than 1 expected." << endl;
    }

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

Exception Handling

- Please note that
  - The throw statement can throw any data type including struct type and class type objects
  - The catch block can catch any data type
  - If we are interested in different types of errors then we can throw different data types from the different parts of a program and then have multiple catch blocks such that each catch block catches different data types
  - This way every throw will be caught by a catch that corresponds the data type of the throw
- See the following example

Exception Handling

- See next slide for the main program

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

bool isPrime(const int x)
{
    //Pre-condition: x is an integer value greater than 1
    //Post-condition: returns true if x is prime and false otherwise
    if (x < 0)
    {
        string msg = "Please don't perform primality test on a negative number.";
        throw(msg);
    }
    if (x < 2)
        throw(x);
    for (int k = 2; k < x; k++)
        if (x % k == 0)
            return false;
    return true;
}
```

Exception Handling

```
int main()
{
    int x;
    srand(time(0));
    x = rand() % 11 - 5;
    try
    {
        if (isPrime(x) == true)
            cout << x  << " is prime." << endl;
        else
            cout << x << " is not prime." << endl;
    }
    catch (int e)
    {
        cout << "An integer greater than 1 expected." << endl;
    }
    catch (string s)
    {
        cout << s << endl;
    }

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

Exception Handling

- Sometimes, we may require a catch block to catch any remaining data type that are not caught in the catch blocks listed
- This is similar to else block in conditional statements
- C++ allows such a catch whose syntax is given as follows

```
catch(…)
```