# Experiment 5
# Classes and Objects - II

## Objectives

To write simple computer programs using classes and objects.

## Prelab Activities

### Programming Output

For each of the given program segments, read the code and write the output in the space provided below each program. [Note: Do not execute these programs on a computer.]

For Programming Output Exercises 1–2, use the following class definition.

```
1  // Time abstract data type (ADT) definition
2  class Time
3  {
4  public:
5     Time(); // constructor
6     Time( int, int, int ); // three-argument constructor
7     void setTime( int, int, int ); // set hour, minute, second
8     void printUniversal(); // print universal time format
9     void printStandard(); // print standard time format
10 private:
11    int hour; // 0 - 23 ( 24-hour clock format )
12    int minute; // 0 - 59
13    int second; // 0 - 59
14 }; // end class Time
15
16 // Time constructor initializes each data member to zero.
17 // Ensures all Time objects start in a consistent state.
18 Time::Time()
19 {
20    hour = minute = second = 0;
21 } // end Time constructor
22
23 // Time constructor initializes each data member as specified.
24 Time::Time( int h, int m, int s )
25 {
26    setTime( h, m, s );
27 } // end Time constructor
28
29 // Set a new Time value using universal time. Perform validity
30 // checks on the data values. Set invalid values to zero.
31 void Time::setTime( int h, int m, int s )
32 {
33    hour = ( h >= 0 && h < 24 ) ? h : 0;
34    minute = ( m >= 0 && m < 60 ) ? m : 0;
35    second = ( s >= 0 && s < 60 ) ? s : 0;
36 } // end function setTime
37
38 // Print Time in universal format
39 void Time::printUniversal()
40 {
41    cout << setfill( '0' ) << setw( 2 ) << hour << ":"
42         << setw( 2 ) << minute << ":"
```

```
43        << setw( 2 ) << second;
44 } // end function printUniversal
45
46 // Print Time in standard format
47 void Time::printStandard()
48 {
49    cout << ( ( hour == 0 || hour == 12 ) ? 12 : hour % 12 )
50         << ":" << setfill( '0' ) << setw( 2 ) << minute
51         << ":" << setw( 2 ) << second
52         << ( hour < 12 ? " AM" : " PM" );
53 } // end function printStandard
```

**1.** What is output by the following code segment? Use the definition of the class Time?

```
1 Time t1();
2
3 t1.setTime( 18, 22, 9 );
4 cout << "The time is: ";
5 t1.printStandard();
```

**2.** What is output by the following main function?

```
1  Time t( 3, 4, 5 );
2
3  t.printStandard();
4  cout << endl;
5
6  t.printUniversal();
7  cout << endl;
8
9  t.setTime( 99, 3, 4);
10
11 t.printUniversal();
12 cout << endl;
```

**3.** What is output by the following program? Use the Time class.

```
1  #include <iostream>
2  using namespace std;
3
4  class M
5  {
6  public:
7     M( int );
8     int mystery( int );
9  private:
10    int data;
11    double number;
12 }; // end class M
13
14 // constructor
15 M::M( int q )
16 {
17    data = q;
18    number = .5;
19 } // end class M constructor
20
```

```
21 // function mystery definition
22 int M::mystery( int q )
23 {
24    data += q;
25    return data * number;
26 } // end function mystery
27
28 int main()
29 {
30    M stuff( 44 );
31    cout << stuff.mystery( 78 );
32 } // end main
```

**4.** What is output by the following program?

```
1  #include <iostream>
2  using namespace std;
3
4  class M
5  {
6  public:
7     M( int );
8     int mystery( int );
9  private:
10    int data;
11    int number;
12
13 }; // end class M
14
15 // constructor
16 M::M( int q = 0 )
17 {
18    data = q;
19    number = 2;
20 } // end class M constructor
21
22 // function mystery definition
23 int M::mystery( int q )
24 {
25    data += q;
26    return data;
27 } // end function mystery
28
29 int main()
30 {
31    M mObject( 2 );
32    M *mPtr = &mObject;
33
34    cout << mObject.mystery( 20 ) << endl;
35    cout << mPtr->mystery( 30 );
36 } // end main
```

**5.** What is output by the following program?

```
1  #include <iostream>
2  using namespace std;
3
4  // class Test definition
```

```
5  class Test
6  {
7  public:
8     Test( int = 0 );
9     void print() const;
10 private:
11    int x;
12 }; // end class Test
13
14 // default constructor
15 Test::Test( int a )
16 {
17    x = a;
18 } // end class Test constructor
19
20 // function print definition
21 void Test::print() const
22 {
23    cout << x
24         << this->x
25         << ( *this ).x << endl;
26 } // end function print
27
28 int main()
29 {
30    Test testObject( 4 );
31
32    testObject.print();
33 } // end main
```

## Correct the Code

For each of the given program segments, determine if there is an error in the code. If there is an error, specify whether it is a logic or compilation error, write the corrected code. If the code does not contain an error, write "no error." For code segments, assume the code appears in main and that using directives are provided. [*Note*: It is possible that a program segment may contain multiple errors.]

**6.** The following should define class Time:

```
1  class Time
2  {
3  public:
4     Time( int = 0, int = 0, int );
5     void setTime( int, int, int );
6     void printUniversal();
7     void printStandard();
8  private:
9     int hour;
10    int minute;
11    int second;
12 } // end class Time
```

**7.** The following code defines class Q:

```
1  class Q
2  {
3  public:
4     int Q( int );
5     void setQ( int );
6     void printQ();
7     int operateQ( int );
8  private:
9     int qData;
10 }; // end class Q
```

**8.** The following is another version of class Q's definition:

```
1  class Q
2  {
3  public:
4    Q( int );
5    void setQ( int );
6    void printQ();
7    int operateQ( int );
8  private:
9    int qData = 1;
10 }; // end class Q
```

**9.** The following defines Q's setQ method. This definition resides outside class Q's definition. Use the corrected class Q from Correct the Code Exercise 11:

```
1 void setQ( int input )
2 {
3    qData = input;
```

```
4 }
```

**10.** The following defines setHour, a member function of the Time class

```
1 int &Time::setHour( int hh )
2 {
3    hour = ( hh >= 0 && hh < 24 ) ? hh : 0;
4
5    return hour;
6 }
```

**11.** The following code should call member function printUniversal of the Time class

```
1 Time clock( 11, 22, 43 );
2 Time *clockPtr = &clock;
3
4 clockPtr.printUniversal();
```

**12.** The following defines class Increment (note the use of const data member increment):

```
1  #include <iostream>
2  using namespace std;
3
4  // class Increment definition
5  class Increment
6  {
7  public:
8     Increment( int c = 0, int i = 1 );
9     void addIncrement() { count += increment; }
10    void print() const;
11 private:
12    int count;
13    const int increment;
14 }; // end class Increment
15
16 // constructor
17 Increment::Increment( int c, int i )
18 {
19    count = c;
20    increment = i;
21 } // end class Increment constructor
22
23 // function print definition
24 void Increment::print() const
25 {
26    cout << "count = " << count
27         << ", increment = " << increment << endl;
28 } // end function print
```

**13.** The code that follows is a definition for class Time and its member functions.

```
1 // class Time definition
2 class Time
3 {
```

```
4  public:
5    Time( int = 0, int = 0, int = 0 );
6
7    void setTime( int, int, int ) const;
8
9    void setHour( int ) const;
10   int getHour() const;
11
12   void setMinute( int ) const;
13   int getMinute() const;
14
15   void setSecond( int ) const;
16   int getSecond() const;
17
18   void printUniversal() const;
19   void printStandard();
20 private:
21   int hour;
22   int minute;
23   int second;
24 }; // end class Time



25 // Member function definitions for Time class.
26 #include <iostream>
27 using namespace std;
28
29 #include "time.h"
30
31 // constructor function to initialize private data
32 // default values are 0 (see class definition)
33 Time::Time( int hr, int min, int sec )
34 {
35    setTime( hr, min, sec );
36 } // end class Time constructor
37
38 // set values of hour, minute and second.
39 void Time::setTime( int h, int m, int s )
40 {
41    setHour( h );
42    setMinute( m );
43    setSecond( s );
44 } // end function setTime
45
46 // set hour value
47 void Time::setHour( int h )
48 {
49    hour = ( h >= 0 && h < 24 ) ? h : 0;
50 } // end function setHour
51
52 // set minute value
53 void Time::setMinute( int m )
54 {
55    minute = ( m >= 0 && m < 60 ) ? m : 0;
56 } // end function setMinute
57
58 // set second value
59 void Time::setSecond( int s )
60 {
61    second = ( s >= 0 && s < 60 ) ? s : 0;
```

```
62 } // end function setSecond
63
64 // get hour value
65 int Time::getHour() const
66 {
67    return hour;
68 } // end functiongetHour
69
70 // get minute value
71 int Time::getMinute() const
72 {
73    return minute;
74 } // end function setMinute
75
76 // get second value
77 int Time::getSecond() const
78 {
79    return second;
80 } // end function getSecond
81
82 // display universal format time: HH:MM
83 void Time::printUniversal() const
84 {
85    cout << ( hour < 10 ? "0" : "" ) << hour << ":"
86         << ( minute < 10 ? "0" : "" ) << minute;
87 } // end function printUniversal
88
89 // display standard format time: HH:MM:SS AM (or PM)
90 void Time::printStandard()
91 {
92    cout << ( ( hour == 12 ) ? 12 : hour % 12 ) << ":"
93         << ( minute < 10 ? "0" : "" ) << minute << ":"
94         << ( second < 10 ? "0" : "" ) << second
95         << ( hour < 12 ? " AM" : " PM" );
96 } // end function printStandard
```

**14.** The code that follows is a definition for class Time. Note the member function that begins a new day by resetting the hour to zero.

```
1  // class Time definition
2  class Time
3  {
4  public:
5     Time( int = 0, int = 0, int = 0 );
6
7     void setTime( int, int, int );
8
9     void setHour( int );
10    int getHour() const;
11
12    void setMinute( int );
13    int getMinute() const;
14
15    void setSecond( int );
16    int getSecond() const;
17
```

```
18     // function newDay definition
19     void newDay() const
20     {
21        setHour( 0 );
22     } // end function newDay
23
24     void printUniversal() const;
25     void printStandard();
26  private:
27     int hour;
28     int minute;
29     int second;
30  }; // end class Time
```

**15.** The following is a definition for class Time:

```
1  // class Time definition
2  class Time
3  {
4  public:
5     Time( int = 0, int = 0, int = 0 ) const;
6
7     void setTime( int, int, int );
8
9     void setHour( int );
10    int getHour() const;
11
12    void setMinute( int );
13    int getMinute() const;
14
15    void setSecond( int );
16    int getSecond() const;
17
18    void printUniversal() const;
19    void printStandard();
20 private:
21    int hour;
22    int minute;
23    int second;
24 }; // end class Time
```

**16.** The code that follows is a definition of the getCount member function. Variable count is a static int that stores the number of objects instantiated. Assume that this definition is located within a class definition.

```
1  static int getCount()
2  {
3     return this->count;
4  }
```

## Lab Exercises

### Lab Exercise 1 — StringBuilder

The problem is divided into six parts:

1. Lab Objectives
2. Description of the Problem
3. UML Diagram
4. Sample Output
5. Test Code
6. Problem-Solving Tips

The program given represent a test application of your class. Read the problem description and examine the sample output. Using the problem-solving tips as a guide, write the StringBuilder class. Compile and execute the program. Compare your output with the sample output provided.

### Lab Objectives

In this lab, you will practice:

- Creating new helper class by writing class definitions.
- Defining member functions of programmer-defined classes.
- Learn member function overloading of a class.
- Learn “stringstream” class and its member functions.
- Instantiating objects from programmer-defined classes.
- Calling member functions of programmer-defined classes.

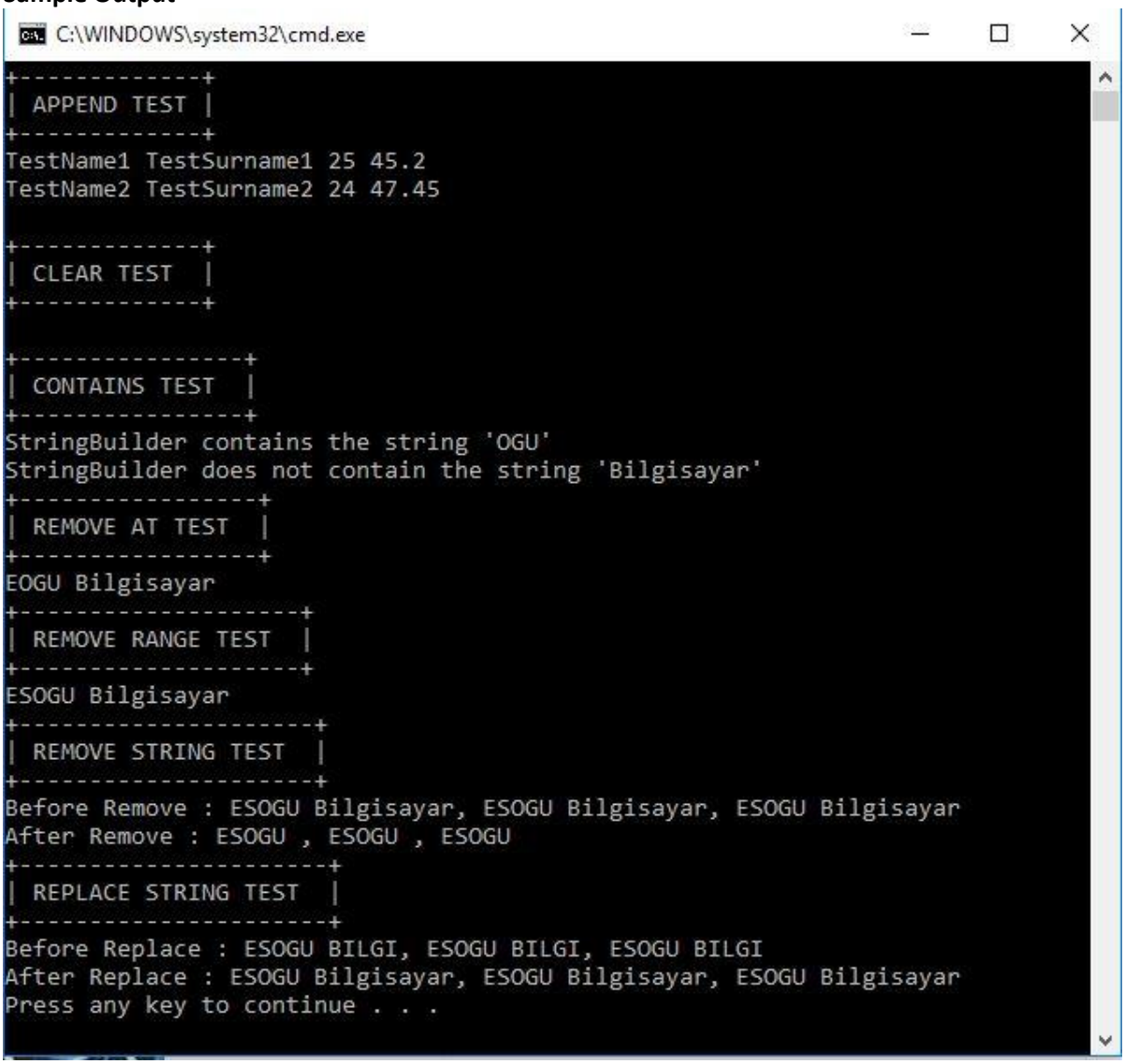
### Description of the Problem

Create a class called StringBuilder to perform string appending, removing operations for some primitive data types, such as int,float,double,char etc. StringBuilder class has member functions and the function prototypes is given to you via UML diagram. The class also has a member variable of type “stringstream” called stream that is encapsulated in private scope of the class. You requirements is to write a header and source file for the class. Implement the member functions and test your implementation by using the given main function. **You are not allowed to change the test code.**

**UML Diagram:**


- Append(…) : All append functions is used to perform string appending operation.
- Clear() : Remove the content of the stream object
- Contains(…) : Check the stream object contents and returns true if given string is located in the stream, otherwise false.
- GetData() : Returns the content of the stream object
- Remove(removeChar) : Removes all characters which is equal to the given character in the stream objects.
- Remove(startIndex,charCount) : Removes characters starting from the given index until given charCount removed.
- RemoveAt(index) : Removes the character at the given index.
- Remove(removeString): Removes all occurrences of the given string from the stream.
- Replace(replacedString, newString): Replaces all occurrences of the given string by the newstring.

**Sample Output**




**Test Code (You are not allowed to change the code)**

```
/// StringBuilderTest.cpp
#include "StringBuilder.h"

void TEST_Append(StringBuilder& sb)
{
	cout << "+-------------+" << endl
		<< "| APPEND TEST |" << endl
		<< "+-------------+" << endl;
	sb.Append("TestName1").Append(" ").Append("TestSurname1")
		.Append(" ").Append(25).Append(" ")
		.Append(45.2).AppendNewLineCharacter();

	sb.Append("TestName2").Append(" ").Append("TestSurname2")
		.Append(" ").Append(24).Append(" ")
		.Append(47.45).AppendNewLineCharacter();
```

```
	cout << sb.GetData() << endl;
	sb.Clear();
}

void TEST_Clear(StringBuilder& sb)
{
	cout << "+-------------+" << endl
		<< "| CLEAR TEST  |" << endl
		<< "+-------------+" << endl;
	sb.Clear();
	cout << sb.GetData() << endl;
}

void TEST_Contains(StringBuilder& sb)
{
	cout << "+----------------+" << endl
		<< "| CONTAINS TEST  |" << endl
		<< "+----------------+" << endl;
	sb.Append("ESOGU");
	if (sb.Contains("OGU"))
	{
		cout << "StringBuilder contains the string 'OGU'" << endl;
	}
	else
	{
		cout << "StringBuilder does not contain the string 'OGU'" << endl;
	}

	if (sb.Contains("Bilgisayar"))
	{
		cout << "StringBuilder contains the string 'Bilgisayar'" << endl;
	}
	else
	{
		cout << "StringBuilder does not contain the string 'Bilgisayar'" <<
endl;
	}

	sb.Clear();
}

void TEST_RemoveAt(StringBuilder& sb)
{
	cout<< "+-----------------+" << endl
		<< "| REMOVE AT TEST  |" << endl
		<< "+-----------------+" << endl;
	sb.Append("ESOGU Bilgisayar");
	sb.RemoveAt(1);
	cout << sb.GetData() << endl;
	sb.Clear();
}

void TEST_RemoveChar(StringBuilder& sb)
{

	cout << "+------------------------+" << endl
		<< "| REMOVE CHARACTER TEST  |" << endl
		<< "+------------------------+" << endl;
	sb.Append("ESOGU Bilgisayar");
	sb.Remove('a');
```

```
	cout << sb.GetData() << endl;
	sb.Clear();
}

void TEST_RemoveRange(StringBuilder& sb)
{
	cout<< "+--------------------+" << endl
		<< "| REMOVE RANGE TEST  |" << endl
		<< "+--------------------+" << endl;
	sb.Append("ESOGU BilgisayarBilgisayar");
	sb.Remove(16, 10);
	cout << sb.GetData() << endl;
	sb.Clear();
}

void TEST_RemoveString(StringBuilder& sb)
{
	cout <<"+----------------------+" << endl
		<< "| REMOVE STRING TEST  |" << endl
		<< "+----------------------+" << endl;
	sb.Append("ESOGU Bilgisayar, ESOGU Bilgisayar, ESOGU Bilgisayar");
	cout << "Before Remove : " << sb.GetData() << endl;
	sb.Remove("Bilgisayar");
	cout << "After Remove : " <<sb.GetData() << endl;
	sb.Clear();
}

void TEST_Replace(StringBuilder& sb)
{
	cout<< "+----------------------+" << endl
		<< "| REPLACE STRING TEST  |" << endl
		<< "+----------------------+" << endl;
	sb.Clear();
	sb.Append("ESOGU BILGI, ESOGU BILGI, ESOGU BILGI");

	cout << "Before Replace : " << sb.GetData() << endl;
	sb.Replace("BILGI", "Bilgisayar");
	cout << "After Replace : " <<sb.GetData() << endl;
	sb.Clear();
}

int main()
{
	StringBuilder strBuilder;
	TEST_Append(strBuilder);
	TEST_Clear(strBuilder);
	TEST_Contains(strBuilder);
	TEST_RemoveAt(strBuilder);
	TEST_RemoveRange(strBuilder);
	TEST_RemoveString(strBuilder);
	TEST_Replace(strBuilder);

	return 0;
}
```

**Problem-Solving Tips**

**1-** Use UML Diagram, and test code.