CS118 – Programming Fundamentals

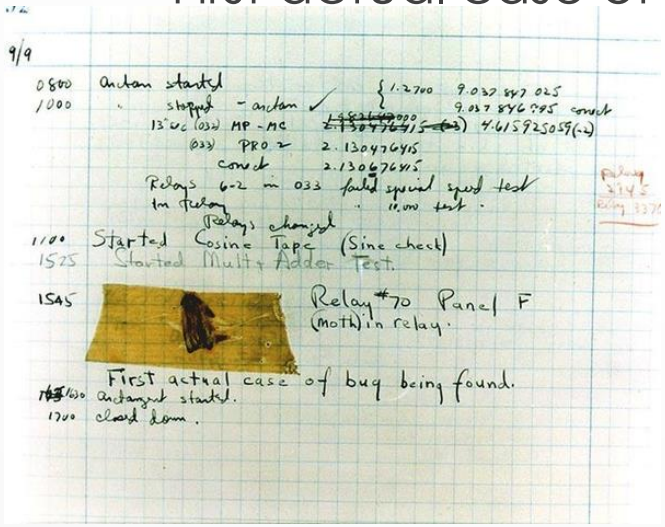
Lecture # 08
Monday, September 16, 2019
FALL 2019
FAST – NUCES, Faisalabad Campus

Zain Iqbal

Testing and Debugging

- Bug
  - A logical mistake in a program
- Debugging
  - Eliminating mistakes in programs
  - Term used when a moth caused a failed relay on the Harvard Mark 11 computer. Grace Hopper and other programmers taped the moth in logbook stating:

"First actual case of a bug being found."



Program Errors

- **Syntax errors**
  - Violation of the grammar rules of the language
  - Discovered by the compiler
  - Error messages may not always show correct location of errors
- **Run-time errors**
  - Error conditions detected by the computer at run-time
- **Logic errors**
  - Errors in the program's algorithm
  - Most difficult to diagnose
  - Computer does not recognize as an error

What makes a bad program?

- Writing Code **without detailed analysis and design**
- Repeating trial and error **without understanding the problem**
- Debugging the program line by line, statement by statement
- **Writing tricky and dirty programs**

Arithmetic Operators and Operator Precedence

C++ arithmetic operators:

- + addition
- - subtraction
- * multiplication
- / division
- % modulus operator
- +, -, *, and / can be used with integral and floating-point data types
- Operators can be **unary** or **binary**

Example 2-3

Arithmetic Expression	Results	Description
5 / 2	2	In the division 5 / 2, the quotient is 2 and the remainder is 1. Therefore, 5 / 2 with the integral operands evaluates to the quotient, which is 2.
14 / 7	2	In the division 14 / 7, the quotient is 2 and remainder is 0. Therefore, 14 / 7 with integral operands evaluates to 2
34 % 5	4	In the division 34 / 5, the quotient is 6 and the remainder is 4. Therefore, 34 %5 with the integral operands evaluates to the remainder, which is 4.
4 % 6	4	In the division 4 / 6, the quotient is 0 and the remainder is 4. Therefore, 4 % 6 with the integral operands evaluates to the remainder, which is 4.

Example

```
#include <iostream>

using namespace std;

int main()
{
    cout << "5.0 + 3.0 = " << 5.0 + 3.0 << endl;
    cout << "3.0 + 9.4 = " << 3.0 + 9.4 << endl;
    cout << "16.3 - 5.2 = " << 16.3 - 5.2 << endl;
    cout << "4.2 * 2.5 = " << 4.2 * 2.5 << endl;
    cout << "5.0 / 2.0 = " << 5.0 / 2.0 << endl;
    cout << "34.5 / 6.0 = " << 34.5 / 6.0 << endl;
    cout << "34.5 / 6.5 = " << 34.5 / 6.5 << endl;

    return 0;
}
```

Sample Run:

```
5.0 + 3.5 = 8.5
3.0 + 9.4 = 12.4
16.3 - 5.2 = 11.1
4.2 * 2.5 = 10.5
5.0 / 2.0 = 2.5
34.5 / 6.0 = 5.75
34.5 / 6.5 = 5.30769
```

C++ Math Operator Rules

- * : Multiplication
- / : Division
  - Integer division truncates remainder
    - 7 / 5 evaluates to 1
- % : Modulus operator returns remainder
  - 7 % 5 evaluates to 2
- + : Addition
- - : Subtraction

Operator	Operation (s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*, / or %	Multiplication, Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ or -	Addition, Subtraction	Evaluated last. If there are several, they are evaluated left to right

Order of Precedence

- All operations inside of () are evaluated first
- *, /, and % are at the same level of precedence and are evaluated next
- + and – have the same level of precedence and are evaluated last
- When operators are on the same level
  - Performed from left to right **(associativity)**
- 3 * 7 - 6 + 2 * 5 / 4 + 6 means

  (((3 * 7) – 6) + ((2 * 5) / 4)) + 6

Example

```
   (((3 * 7) - 6) + ((2 * 5) / 4 )) + 6
= ((21 - 6) + (10 / 4)) + 6        (Evaluate *)
= ((21 - 6) + 2) + 6               (Evaluate /.  Note that this is an integer division.)
= (15 + 2) + 6                     (Evaluate –)
= 17 + 6                           (Evaluate first +)
= 23                               (Evaluate +)
```

Example

In the expression:

```
3 + 4 * 5
```

* is evaluated before +. Therefore, the result of this expression is 23. On the other hand, in the expression:

```
(3 + 4) * 5
```

+ is evaluated before * and the result of this expression is 35.

Expressions

- If all operands are integers
  - Expression is called an integral expression
    - Yields an integral result
    - Example: 2 + 3 * 5
- If all operands are floating-point
  - Expression is called a floating-point expression
    - Yields a floating-point result
    - Example: 12.8 * 17.5 - 34.50

Examples

- Consider the following C++ integral expressions:
  - 2 + 3 * 5
  - 3 + x - y / 7
  - x + 2 * (y - z) + 18
- In these expressions, x, y, and z represent variables of the integer type; that is, they can hold integer values

Examples

- Consider the following C++ floating-point expressions:
  - 12.8 * 17.5 - 34.50
  - x * 10.5 + y - 16.2
- Here, x and y represent variables of the floating-point type; that is, they can hold floating-point values

Mixed Expressions

- Mixed expression:
  - Has operands of different data types
  - Contains integers and floating-point
- Examples of mixed expressions:
  - 2 + 3.5
  - 6 / 4 + 3.9
  - 5.4 * 2 – 13.6 + 18 / 2

Mixed Expressions (cont'd.)

- Evaluation rules:

  Rule # 1
  - If operator has same types of operands
  - Evaluated according to the type of the operands
  - If operator has both types of operands
  - Integer is changed to floating-point with decimal part "0"
  - Operator is evaluated
  - Result is floating-point

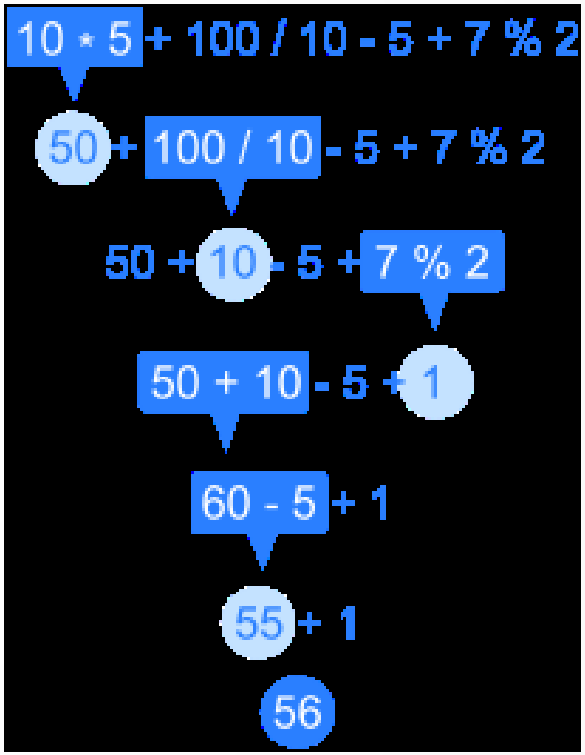
  Rule # 2
  - Entire expression is evaluated according to precedence rules

Examples

Mixed Expression	Evaluation	Rule Applied
`3/2+5.5`	`=1+5.5`	`3/2=1` (integer division; Rule 1(a))
	`=6.5`	`(1+5.5`
		`=1.0+5.5` (Rule 1(b))
		`=6.5)`
`15.6/2+5`	`=7.8+5`	`15.6/2`
		`=15.6/2.0` (Rule 1(b))
		`=7.8`
	`=12.8`	`7.8+5`
		`=7.8+5.0` (Rule1(b))
		`=12.8`
`4+5/2.0`	`=4+2.5`	`5/2.0=5.0/2.0` (Rule1(b))
		`=2.5`
	`=6.5`	`4+2.5=4.0+2.5` (Rule1(b))
		`=6.5`
`4*3+7/5-25.5`	`=12+7/5-25.5`	`4*3=12;` (Rule 1(a))
	`=12+1-25.5`	`7/5=1` (integer division; Rule 1(a))
	`=13-25.5`	`12+1=13;` (Rule 1(a))
	`=-12.5`	`13-25.5=13.0-25.5` (Rule 1(b))
		`=-12.5`

Example

- Solve this expression on notebook
- 10 * 5 + 100/10 – 5 + 7 % 2



Example

```
// This program illustrates how mixed expressions are evaluated.

#include <iostream>

using namespace std;

int main()
{
     cout << "3 / 2 + 5.5 = " << 3 / 2 + 5.5 << endl;
     cout << "15.6 / 2 + 5 = " << 15.6 / 2 + 5 << endl;
     cout << "4 + 5 / 2.0 = " << 4 + 5 / 2.0 << endl;
     cout << "4 * 3 + 7 / 5 - 25.5 = "
          << 4 * 3 + 7 / 5 - 25.5
          << endl;

     return 0;
}
```

Example (Integers)

2 + 5 = 7

13 + 89 = 102

34 - 20 = 14

45 - 90 = -45

2 * 7 = 14

5 / 2 = 2

14 / 7 = 2

34 % 5 = 4

4 % 6 = 4

Example (Float)

5.0 + 3.5 = 8.5

3.0 + 9.4 = 12.4

16.3 - 5.2 = 11.1

4.2 * 2.5 = 10.5

5.0 / 2.0 = 2.5

34.5 / 6.0 = 5.75

34.5 / 6.5 = 5.30769

Type Conversion (Casting)

- **Implicit type coercion:** When value of one type is automatically changed to another type
  - 2 + 3.4 = 2.0 + 3.4 = 5.4
  - int num1;
  - float num2 = 12.45;
  - num1 = num2;
  - num1 = ?
- **cast operator:** provides explicit type conversion

  static_cast<dataTypeName>(expression)

Type Conversion (cont'd.)

Expression	Evaluates to
`static_cast<int>(7.9)`	7
`static_cast<int>(3.3)`	3
`static_cast<double>(25)`	25.0
`static_cast<double>(5+3)`	= `static_cast<double>(8)` = 8.0
`static_cast<double>(15) / 2`	= 15.0 / 2 (because `static_cast<double>(15)` = 15.0) = 15.0 / 2.0 = 7.5
`static_cast<double>(15 / 2)`	= `static_cast<double>(7)` (because 15 / 2 = 7) = 7.0
`static_cast<int>(7.8 +` `static_cast<double>(15) / 2)`	= `static_cast<int>(7.8+7.5)` = `static_cast<int>(15.3)` = 15
`static_cast<int>(7.8 +` `static_cast<double>(15 / 2))`	= `static_cast<int>(7.8 + 7.0)` = `static_cast<int>(14.8)` = 14

Type Conversion (cont'd.)

- You can also use cast operators to explicitly convert
- **char** data values into **int** data values
- **int** data values into **char** data values
- To convert char data values into int data values, you use a collating sequence
- For example, in the ASCII character set
- static_cast<int>('A') is 65 and static_cast<int>('8') is 56
- Similarly, static_cast<char>(65) is 'A' and static_cast<char>(56) is '8'

string **Type**

- Programmer-defined type supplied in ANSI/ISO Standard C++ library
- Sequence of zero or more characters
- Enclosed in double quotation marks
- **Null:** a string with no characters
- Each character has relative position in string
  - Position of first character is 0
- Length of a string is number of characters in it
  - Example: length of "William Jacob" is 13

Using the string Data Type in a Program

- To use the string type, you need to access its definition from the header file string
- Include the following preprocessor directive:

```
#include  <string>
```

Using string **datatype**

String	Position of a Character in the String	Length of String
"Test String"	Position of 'T' is 0 Position of 'i' is 8 Position of ' ' (the space) is 4 Position of 'S' is 5 Position of 'g' is 10	11
"String"	Position of 'S' is 0 Position of 't' is 1 Position of 'r' is 2 Position of 'i' is 3 Position of 'n' is 4 Position of 'g' is 5	6

When determining the length of a string, you must also count any spaces in the string. For example, the length of the following string is 22

"It is a beautiful day."

Allocating Memory with Constants **and** Variables

- Storing data in the computer's memory is a two-step process:
  1. Instruct the computer to allocate memory
  2. Include statements in the program to put data into the allocated memory

Contd..

- **Named constant:** A memory location whose content is not allowed to change during program execution.
- To allocate memory, we use C++'s declaration statements. The syntax to declare a named constant is:

```
const dataType identifier = value;
```

Consider the following C++ statements:

```
const double CONVERSION = 2.54;
const int NO_OF_STUDENTS = 20;
const char BLANK = ' ';
```

Not Warn for mistyped value

Variable

- **Variable:** A memory location whose content may change during program execution. The syntax for declaring one variable or multiple variables is:

```
dataType identifier, identifier, . . .;
```

Simple data type Const/Var can hold one value at a time

Consider the following statements:

```
double amountDue;
int counter;
char ch;
int x, y;
string name;
```

Assignment Statement

- The assignment statement takes the form:

```
variable = expression;
```

Should match datatype of variable

- Expression is evaluated and its value is assigned to the variable on the left side
- In C++, '**=**' is called the assignment operator

- Value can be assigned by taking input from user

Assignment Statement (cont'd.)

Example 1:

```
int num1, num2;
double sales;
char ch;
float average;
string str;

num1 = 4;
num2 = 4 * 5 - 10 ;
sale = 0.03 * 50000;
ch = 'A';
str = "It is sunny day";
```

==========

Example 2:

1. num1 = 18 ;
2. num1 = num1 + 27 ;
3. num2 = num1 ;
4. num3 = num2 / 5 ;
5. num3 = num3 / 4 ;

Variable and memory

1. num1 = 18 ;
2. num1 = num1 + 27 ;
3. num2 = num1 ;
4. num3 = num2 / 5 ;
5. num3 = num3 / 4 ;

	Values of the Variables			Explanation
Before Statement 1	? num1	? num2	? num3	
After Statement 1	18 num1	? num2	? num3	
After Statement 2	45 num1	? num2	? num3	num1 + 27 = 18 + 27 = 45. This value is assigned to num1, which replaces the old value of num1.
After Statement 3	45 num1	45 num2	? num3	Copy the value of num1 into num2.
After Statement 4	45 num1	45 num2	9 num3	num2 / 5 = 45 / 5 = 9. This value is assigned to num3. So num3 = 9.
After Statement 5	45 num1	45 num2	2 num3	num3 / 4 = 9 / 4 = 2. This value is assigned to num3, which replaces the old value of num3.

Saving and Using the Value of an Expression

- To save the value of an expression:
  - Declare a variable of the appropriate data type
  - Assign the value of the expression to the variable that was declared
    - Use the assignment statement
- Wherever the value of the expression is needed, use the variable holding the value

Declaring & Initializing Variables

- Variables can be initialized when declared:

```
int first=13, second=10;
char ch=' ';
double x=12.6;
```

- All variables must be initialized before they are used
- But not necessarily during declaration

Input Statement

Putting data into variables from the standard input device is accomplished via the use of `cin` and the operator `>>`. The syntax of `cin` together with `>>` is:

```
cin >> variable >> variable ...;
```

This is called an **input (read)** statement. In C++, `>>` is called the **stream extraction operator**.

Suppose that `miles` is a variable of type `double`. Further suppose that the input is `73.65`. Consider the following statements:

```
cin >> miles;
```

This statement causes the computer to get the input, which is `73.65`, from the standard input device and stores it in the variable `miles`. That is, after this statement executes, the value of the variable `miles` is `73.65`.

Two ways to initialize a variable

Consider the following two sets of code:

(a)
```
feet = 35;
inches = 6;
cout << "Total inches = "
     << 12 * feet + inches;
```

(b)
```
cout << "Enter feet: ";
cin >> feet;
cout << endl;
cout << "Enter inches: ";
cin >> inches;
cout << endl;
cout << "Total inches = "
     << 12 * feet + inches;
```

Example


```
int firstNum, secondNum;
char ch;
float z;
string name;
```

```
1.  firstNum = 4;
2.  secondNum = 2 * firstNum + 6;
3.  z = (firstNum + 1) / 2.0;
4.  ch = 'A';
5.  cin >> secondNum;
6.  cin >> z;
7.  firstNum = 2 * secondNum + static_cast<int>(z);
8.  cin >> name;
9.  secondNum = secondNum + 1;
10. cin >> ch;
11. firstNum = firstNum + static_cast<int>(ch);
12. z = firstNum - z;
```

After St.	Values of the Variables					Explanation
1	4 firstNum	? secondNum	? z	? ch	? name	Store 4 into firstNum
2	4 firstNum	14 secondNum	? z	? ch	? name	2 * firstNum + 6 = 2 * 4 + 6 = 14. Store 14 into secondNum.
3	4 firstNum	14 secondNum	2.5 z	? ch	? name	(firstNum + 1) / 2.0 = (4 + 1) / 2.0 = 5 / 2.0 = 2.5. Store 2.5 into z.
4	4 firstNum	14 secondNum	2.5 z	A ch	? name	Store 'A' into ch.
5	4 firstNum	8 secondNum	2.5 z	A ch	? name	Read a number from the keyboard (which is 8) and store it into secondNum. This statement replaces the old value of secondNum with this new value.
6	4 firstNum	8 secondNum	16.3 z	A ch	? name	Read a number from the keyboard (which is 16.3) and store this number into z. This statement replaces the old value of z with this new value.
7	32 firstNum	8 secondNum	16.3 z	A ch	? name	2 * secondNum + static_cast<int>(z) = 2 * 8 + static_cast<int> (16.3) =16 + 16 = 32. Store 32 into firstNum. This statement replaces the old value of firstNum with this new value.

Cont.

8		Read the next input, **Jenny**, from the keyboard and store it into **name**.
9		**secondNum + 1 = 8 + 1 = 9.** Store **9** into **secondNum**.
10		Read the next input from the keyboard (which is **D**) and store it into **ch**. This statement replaces the old value of **ch** with the new value.
11		**firstNum + static_cast<int>(ch) = 32 + static_cast<int> ('D') = 32 + 68 = 100.** Store **100** into **firstNum**.
12		**firstNum – z = 100 – 16.3 = 100.0 – 16.3 = 83.7.** Store **83.7** into **z**.

Output

- In C++, output on the standard output device is accomplished via the use of cout and the operator <<.
- The general syntax of cout together with << is:

```
cout << expression or manipulator << expression or manipulator...;
```

- This is called an output statement. In C++, << is called the stream insertion operator. **[This operator (<<) applied to an output stream is known as insertion operator. because inserts data into an output stream]**
- Generating output with cout follows two rules:

1. The expression is evaluated, and its value is printed at the current insertion point on the output device.
2. A manipulator is used to format the output. The simplest manipulator is endl

Practice of Cout<<

What is the output?

```
int a, b;
a = 65;                          //Line 1
b = 78;                          //Line 2
cout << 29 / 4 << endl;          //Line 3
cout << 3.0 / 2 << endl;         //Line 4
cout << "Hello there.\n";        //Line 5
cout << 7 << endl;               //Line 6
cout << 3 + 5 << endl;           //Line 7
cout << "3 + 5";                 //Line 8
cout << endl;                    //Line 9
cout << a << endl;               //Line 10
cout << "A" << endl;             //Line 11
cout << (a + 5) * 6 << endl;     //Line 12
cout << 2 * b << endl;           //Line 13
```

Output of Statement at

```
7                   //Line 3
1.5                 //Line 4
Hello there.        // Line 5
7                   //Line 6
8                   //Line 7
3 + 5               //Line 8
65                  //Line 10
A                   //Line 11
420                 //Line 12
156                 // Line 13
```

Questions