Programming Principles (MT162)

Lecture 3

Dr. Ahmed Fathalla

I/O Streams and Standard I/O Devices

- **Stream**: sequence of characters from source to destination
- **I/O**: sequence of bytes (stream of bytes) from source to destination
  - Bytes are usually characters, unless program requires other types of information
- **Input stream**: sequence of characters from an **input device** to the **computer**
- **Output stream**: sequence of characters from the **computer** to an **output device**

I/O Streams and Standard I/O Devices (continued)

- Use `iostream` header file to **extract** data from keyboard and **send** output to the screen
  - Contains definitions of two data types:
    - `istream` - input stream
    - `ostream` - output stream
  - Has two variables:
    - `cin` - stands for common input
    - `cout` - stands for common output

cin and the Extraction Operator >>

- The syntax of an input statement using `cin` and the extraction operator >> is:

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

- The extraction operator >> is “binary operator”
  - Left-side **operand** is an input stream variable
    - Example: `cin`
  - Right-side **operand** is a variable

cin and the Extraction Operator >> (continued)

- No difference between a single `cin` with multiple variables and multiple `cin` statements with one variable
- When scanning, `>>` skips all whitespace
  - Blanks and certain nonprintable characters
- `>>` distinguishes between character `2` and number `2` by the right-side operand of `>>`
  - If type `char` or `int` (or `double`), the 2 is treated as a character or as a number `2`

`cin` and the Extraction Operator >> (continued)

EXAMPLE 3-1

```
int a, b;
double z;
char ch, ch1, ch2;
```

	Statement	Input	Value Stored in Memory
1	`cin >> ch;`	`A`	`ch = 'A'`
2	`cin >> ch;`	`AB`	`ch = 'A'`, `'B'` is held for later input
3	`cin >> a;`	`48`	`a = 48`
4	`cin >> a;`	`46.35`	`a = 46`, `.35` is held for later input
5	`cin >> z;`	`74.35`	`z = 74.35`
6	`cin >> z;`	`39`	`z = 39.0`
7	`cin >> z >> a;`	`65.78 38`	`z = 65.78, a = 38`
8	`cin >> a >> b;`	`4 60`	`a = 4, b = 60`
9	`cin >> a >> ch >> z;`	`57 A 26.9`	`a = 57, ch = 'A', z = 26.9`
10	`cin >> a >> ch >> z;`	`57 A` `26.9`	`a = 57, ch = 'A',` `z = 26.9`

EXAMPLE 3-1

```
int a, b;
double z;
char ch, ch1, ch2;
```

11	`cin >> a >> ch >> z;`	57 A 26.9	a = 57, ch = 'A', z = 26.9
12	`cin >> a >> ch >> z;`	57A26.9	a = 57, ch = 'A', z = 26.9
13	`cin >> z >> ch >> a;`	36.78B34	z = 36.78, ch = 'B', a = 34
14	`cin >> z >> ch >> a;`	36.78 B34	z = 36.78, ch = 'B', a = 34
15	`cin >> a >> b >> z;`	11 34	a = 11, b = 34, computer waits for the next number
16	`cin >> a >> z;`	46 32.4 68	a = 46, z = 32.4, 68 is held for later input
17	`cin >> a >> z;`	78.49	a = 78, z = 0.49
18	`cin >> ch >> a;`	256	ch = '2', a = 56
19	`cin >> a >> ch;`	256	a = 256, computer waits for the input value for ch
20	`cin >> ch1 >> ch2;`	A B	ch1 = 'A', ch2 = 'B'

Exercise_1

- Ask the user to input two numbers, then print the sum of the two numbers.

```
Enter two numbers:
9 6
The sum of 9 and 6 is 15
```

Solution

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

int main()
{
   int a,b;
   cout<<"Enter two numbers: \n";
   cin>>a>>b;
   cout<<"The sum of "<<a<<" and "<<b<<" is "<<a+b;
   return 0;
}
```

Exercise_2

- Write a program that takes an integer as the number of minutes and outputs the total hours and minutes (e.g., 90 minutes = 1 hour 30 minutes).

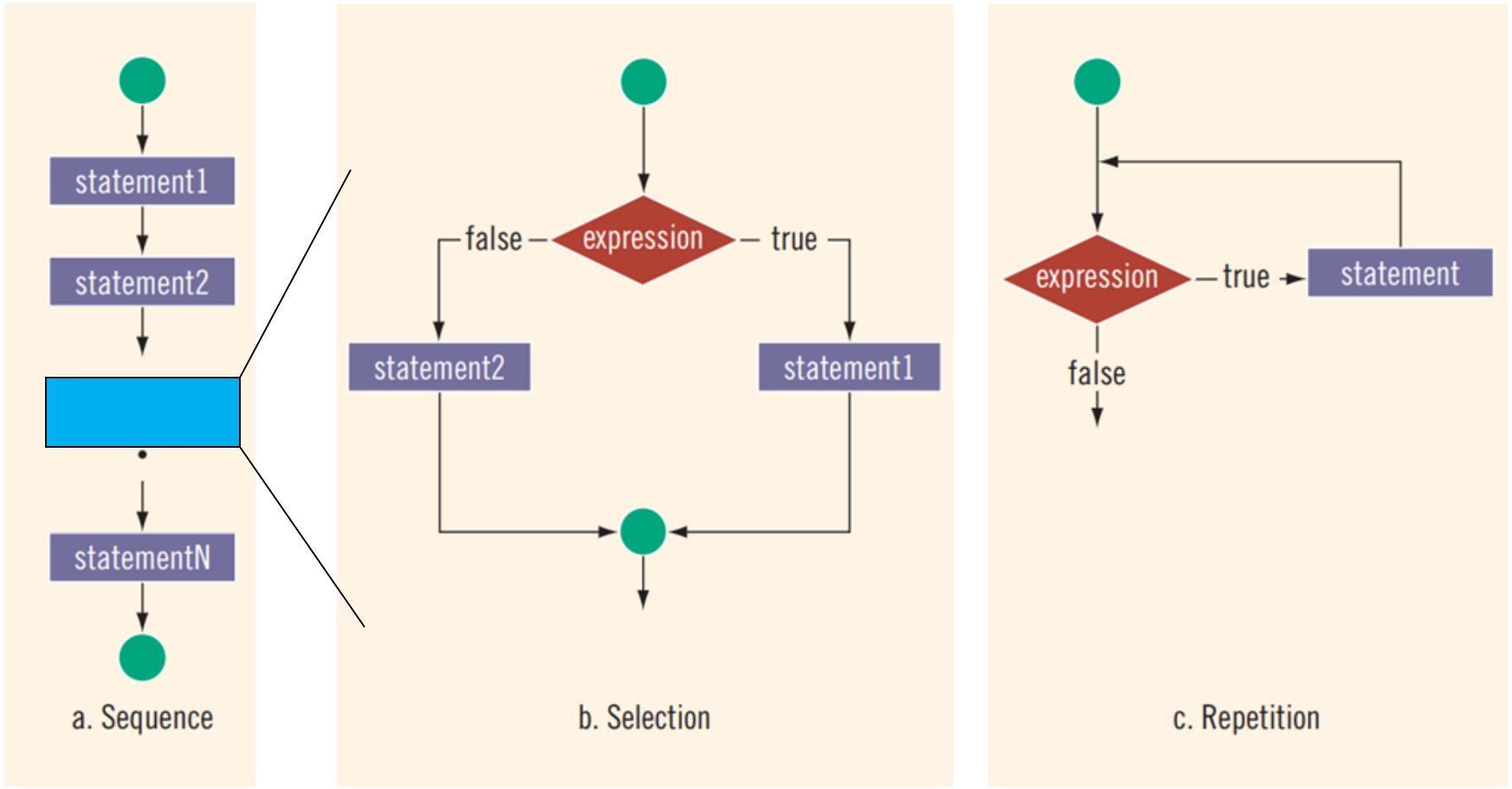
```
Enter number of minutes
150
2 hours, and 30 minutes
```

Solution

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

int main()
{
   int minutes;
   cout<<"Enter number of minutes \n";
   cin>>minutes;
   cout<<minutes/60<<" hours, and "<<minutes%60<<" minutes";
   return 0;
}
```

Control Structures



Control Structures

- A computer can proceed:
  - In sequence
  - Selectively (branch) - making a choice
  - Repetitively (iteratively) - looping
- Some statements are executed **Only If** certain **conditions** are met.
- A **condition** is met if it evaluates to `true.`

Logical expression

- Logical expression: An expression that evaluates to **`true`** or **`false`** is called a logical expression.
- For example, because "8 is greater than 3" is true, the expression 8 > 3 is a **logical expression**.
- Note that '>' is an operator in C+ +, called the "greater than" and is an example of a **relational operator**.

Relational Operators

- The following Table lists the C+ + relational operators.

- Relational operators:
  - Allow comparisons
  - Require two operands (binary)
  - Evaluate to `true` or `false`

Operator	Description
(==)	equal to
(!=)	not equal to
(<)	less than
(<=)	less than or equal to
(>)	greater than
(>=)	greater than or equal to

Relational Operators

EXAMPLE 4-1

Expression	Meaning	Value
`8 < 15`	`8` is less than `15`	`true`
`6 != 6`	`6` is not equal to `6`	`false`
`2.5 > 5.8`	`2.5` is greater than `5.8`	`false`
`5.9 <= 7.5`	`5.9` is less than or equal to `7.5`	`true`
`7 <= 10.4`	`7` is less than or equal to `10.4`	`true`

Logical Operators

- There are situations when the logical expression is a combination of **two or more logical expressions**. For example, suppose weight and height are double variables.

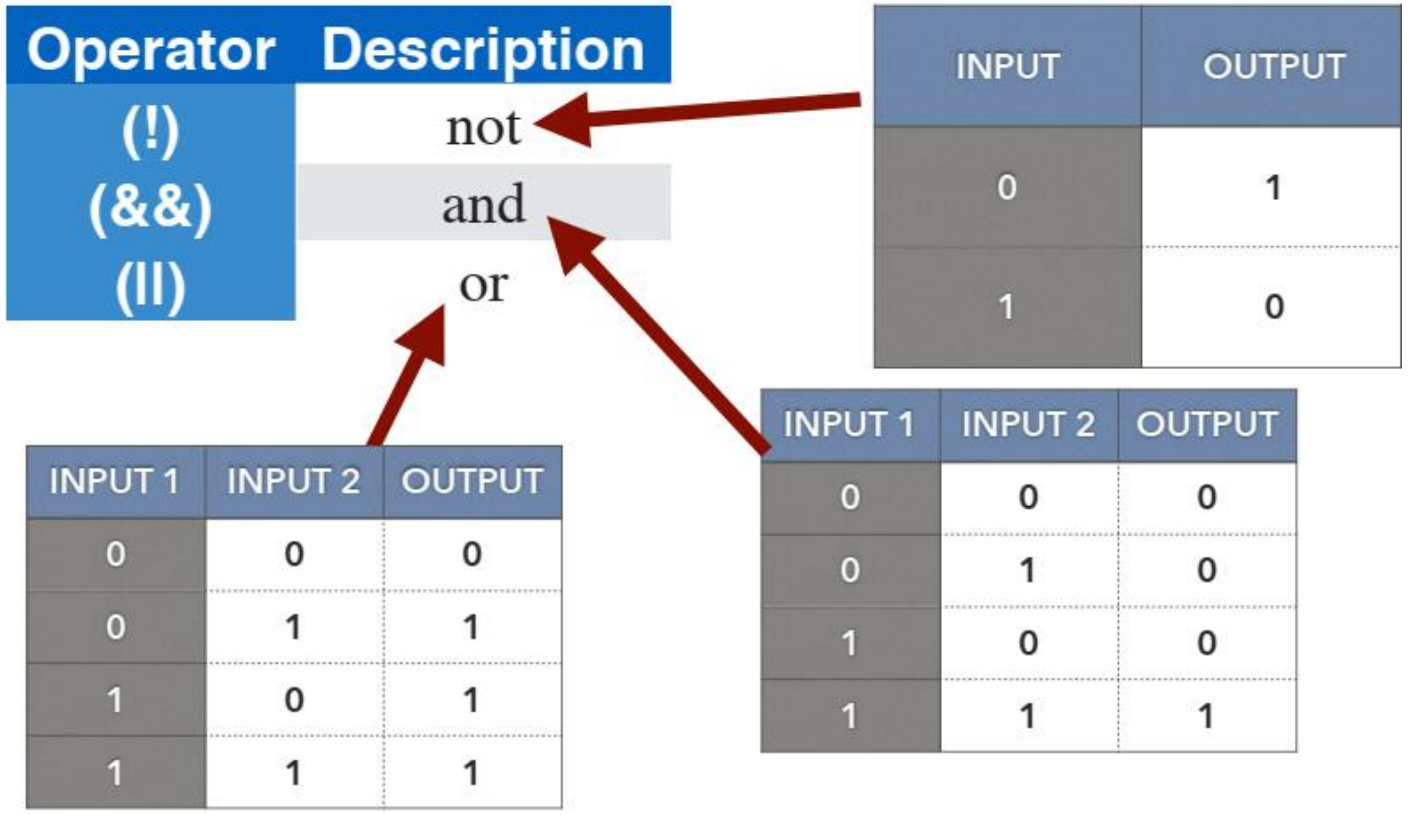
  Consider the following logical expression:

```
weight > 180  and  height < 6.0
```

- The above logical expression is a combination of two logical expressions; weight > 180, and height < 6 . 0, and these logical expressions are combined using the word "and’.

Logical Operators (two or more logical expressions)

Logical (Boolean) operators enable you to combine logical expressions.



Logical (Boolean) Operators and Logical Expressions (continued)

Expression	!(Expression)
`true` (nonzero)	`false` (0)
`false` (0)	`true` (1)

EXAMPLE 4-2

Expression	Value	Explanation
`!('A' > 'B')`	`true`	Because `'A' > 'B'` is `false`, `!('A' > 'B')` is `true`.
`!(6 <= 7)`	`false`	Because `6 <= 7` is `true`, `!(6 <= 7)` is `false`.

Expression1	Expression2	Expression1 && Expression2
`true` (nonzero)	`true` (nonzero)	`true` (1)
`true` (nonzero)	`false` (0)	`false` (0)
`false` (0)	`true` (nonzero)	`false` (0)
`false` (0)	`false` (0)	`false` (0)

Expression	Value	Explanation
`(14 >= 5) && ('A' < 'B')`	`true`	Because `(14 >= 5)` is `true`, `('A' < 'B')` is `true`, and `true && true` is `true`, the expression evaluates to `true`.
`(24 >= 35) && ('A' < 'B')`	`false`	Because `(24 >= 35)` is `false`, `('A' < 'B')` is `true`, and `false && true` is `false`, the expression evaluates to `false`.

Expression1	Expression2	Expression1 || Expression2
true (nonzero)	true (nonzero)	true (1)
true (nonzero)	false (0)	true (1)
false (0)	true (nonzero)	true (1)
false (0)	false (0)	false (0)

Expression	Value	Explanation
`(14>=5) || ('A'>'B')`	true	Because `(14 >= 5)` is true, `('A' > 'B')` is false, and true || false is true, the expression evaluates to true.
`(24>=35) || ('A'>'B')`	false	Because `(24 >= 35)` is false, `('A' > 'B')` is false, and false || false is false, the expression evaluates to false.
`('A'<='a') || (7!=7)`	true	Because `('A' <= 'a')` is true, `(7 != 7)` is false, and true || false is true, the expression evaluates to true.

Expressions

- An expression is a sequence of ***operators*** and their ***operands***, that specifies a computation.
- A C++ program can contain various types of expressions such as arithmetic and strings. For example, length + width is an arithmetic expression.
- Arithmetic expressions are **evaluated** according to rules of arithmetic operations.

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`

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 (continued)

- Evaluation rules:
  - 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
    - Operator is evaluated
    - Result is floating-point
  - Entire expression is evaluated according to precedence rules

Operator Precedence

- Expression:

  Example: `2 + 3 * 5`

- 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.

Operator Precedence

```
3 * 7 - 6 + 2 * 5 / 4 + 6
```

- When operators are on the same level
  - Performed from left to right (associativity)

```
(((3 * 7) - 6) + ((2 * 5) / 4 )) + 6 = 23
```

Operator Precedence

Operators	Precedence
!, +, – (unary operators)	first
*, /, %	second
+, –	third
<, <=, >=, >	fourth
==, !=	fifth
&&	sixth
||	seventh
= (assignment operator)	last

Order of Precedence (continued)

EXAMPLE 4-5

Suppose you have the following declarations:

```
bool found = true;
bool flag = false;
int num = 1;
double x = 5.2;
double y = 3.4;
int a = 5, b = 8;
int n = 20;
char ch = 'B';
```

Order of Precedence (continued)

Expression	Value	Explanation
`!found`	`false`	Because `found` is `true`, `!found` is `false`.
`x > 4.0`	`true`	Because `x` is `5.2` and `5.2 > 4.0` is `true`, the expression `x > 4.0` evaluates to `true`.
`!num`	`false`	Because `num` is `1`, which is nonzero, `num` is `true` and so `!num` is `false`.
`!found && (x >= 0)`	`false`	In this expression, `!found` is `false`. Also, because `x` is `5.2` and `5.2 >= 0` is `true`, `x >= 0` is `true`. Therefore, the value of the expression `!found && (x >= 0)` is `false && true`, which evaluates to `false`.
`!(found && (x >= 0))`	`false`	In this expression, `found && (x >= 0)` is `true && true`, which evaluates to `true`. Therefore, the value of the expression `!(found && (x >= 0))` is `!true`, which evaluates to `false`.
`x + y <= 20.5`	`true`	Because `x + y = 5.2 + 3.4 = 8.6` and `8.6 <= 20.5`, it follows that `x + y <= 20.5` evaluates to `true`.

Order of Precedence (continued)

Expression	Value	Explanation
`(n >= 0) && (n <= 100)`	`true`	Here `n` is 20. Because `20 >= 0` is `true`, `n >= 0` is `true`. Also, because `20 <= 100` is `true`, `n <= 100` is `true`. Therefore, the value of the expression `(n >= 0) && (n <= 100)` is `true && true`, which evaluates to `true`.
`('A' <= ch && ch <= 'Z')`	`true`	In this expression, the value of `ch` is `'B'`. Because `'A' <= 'B'` is `true`, `'A' <= ch` evaluates to `true`. Also, because `'B' <= 'Z'` is `true`, `ch <= 'Z'` evaluates to `true`. Therefore, the value of the expression `('A' <= ch && ch <= 'Z')` is `true && true`, which evaluates to `true`.
`(a + 2 <= b) && !flag`	`true`	Now `a + 2 = 5 + 2 = 7` and `b` is 8. Because `7 <= 8` is `true`, the expression `a + 2 <= b` evaluates to `true`. Also, because `flag` is `false`, `!flag` is `true`. Therefore, the value of the expression `(a + 2 <= b) && !flag` is `true && true`, which evaluates to `true`.