Computer Programing [Lab]

Lab#01: Control Structures

Agenda

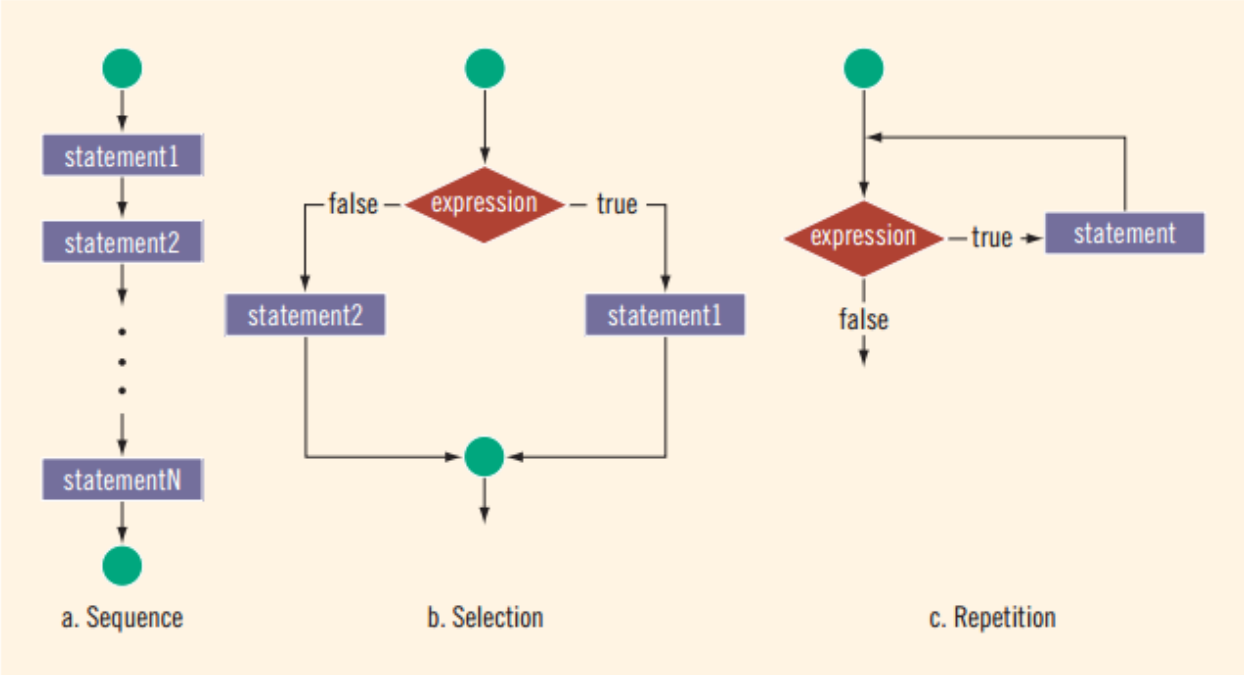
- Recall Previous Concepts
- Control Structures
- If else statements
- Switch statements

Previous Concepts

- How to design a program
- Header file
- Libraries
- Variables
- Reserved words
- Data types
- Arithmetic operators, Logical operators, Relational operators
- Order of Precedence (unary operators, binary operators)
- Stream Insertion operator, Stream Extraction operator (<< , >>)
- cin.get, cin.peek, cin.putback, cin.ignore, setw(), setfill()

Control Structures

Provide alternatives to sequential program execution and are used to alter the sequential flow of execution. The two most common control structures are selection and repetition. In selection, the program executes particular statements depending on some condition(s). In repetition, the program repeats particular statements a certain number of times based on some condition(s).



Boolean Expressions Relational Operators

In C++ the testing of conditions is done using Boolean expressions, which yield bool values that are either true or false. The simplest and most common way to construct such an expression is to use the so-called relational operators.

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

Be careful to avoid mixed-type comparisons. If x is a floating point number and y is an integer the equality tests will not work as expected.

Compound Boolean expressions using logical operators

If you need to test more than one relational expression at a time, it is possible to combine the

Relational expressions using the logical operators.

Operator	Description
!	not
&&	and
||	or

The Simple if Statement

Use

To specify the conditions under which a statement or group of statements should be executed.

Form

```
if (boolean-expression)
        statement;
```

Where IF is a reserved word, boolean-expression is an expression that evaluates to true or false, and statement is a C++ statement or a group of statements enclosed in curly braces (a compound statement).

Action

If the boolean expression is true, the specified statement is executed; otherwise it is not. In either case, execution continues with the next statement in the program.

Examples

```
// grade is a char
if (grade == 'A')
        cout << "EXCELLENT WORK!" << endl;
--------------------------------------------------------------------------
// yards is an int
if (yards != 0)
        cout << "There were " << yards << " yards." << endl;
--------------------------------------------------------------------------
// temperature, normalTemp and degreesOfFever are doubles;
// hasFever is a bool
if (temperature > normalTemp) {
        degreesOfFever = temperature - normalTemp;
        hasFever = true;
}
```

The if-else Statement

Use

To choose exactly one out of two statements (possibly compound statements) to be executed; specifies the conditions under which the first statement is to be executed and provides an alternative statement to execute if these conditions are not met.

Form

```
if (boolean-expression)
        statement-1;
else
        statement-2;
```

Where if and else are reserved words, boolean-expression is an expression that evaluates to true or false, and statement-1 and statement-2 are C++ statements (possibly compound statements, i.e. a group of statements enclosed by curly braces).

Action

If the boolean expression is true, statement-1 is executed and statement-2 is skipped; otherwise statement-1 is skipped and statement-2 is executed. In either case, execution continues with the next statement in the program.

Examples

```
// numItems is an int; averageCost and totalCost are doubles
 if (numItems >= 0)
        averageCost = totalCost / numItems;
else
        cout << "No items were purchased." << endl;
 ---------------------------------------------------------------------
const double POLLUTION_CUTOFF = 3.5;
// pollutionIndexis a double
if (pollutionIndex < POLLUTION_CUTOFF)
        cout << "Safe Condition" << endl;
else
        cout << "Hazardous Condition" << endl;
```

The Nested if-else Statements

Use

To choose one statement (possibly compound) to be executed from among a group of statements (possibly compound); specifies the conditions under which each statement may be executed and may contain a default statement (in an else clause at the end) to be executed if none of these conditions are met. Note that in the absence of a final else clause, it may be the case that none of the statements are executed.

Form

```
if (boolean-expression-1)
        statement-1;
else if (boolean-expression-2)
        statement-2;
        .
        .
        .
else if (boolean-expression-n)
        statement-n;
else
        statement-default;
```

where if and else are reserved words, boolean-expression-1, boolean-expression-2, . . ., boolean-expression-n are expressions that evaluate to true or false, and statement-1, statement-2, . . ., statement-n, and statement-default are C++ statements, possibly compound statements. (A compound statement is a group of statements enclosed by curly braces.)

Action

The boolean expressions are evaluated in the order of their appearance to determine the first expression that is true. The associated statement is executed, and execution continues with the first statement following the entire if-else-if construct. If none of the boolean expressions is true, the statement associated with the else clause is executed, and execution then continues with the statement following the construct. If none of the boolean expressions is true and the else clause is omitted, execution “falls through” to (continues with) the next statement in the program after the construct.

(Next Page)

Examples

```
// score is a double; grade is a char
 if (score == 100) {
        grade = ’A’;
        cout << "Superb" << endl;
}
else if (score >= 90) {
        grade = ’A’;
        cout << "Excellent" << endl;
}
else if (score >= 80) {
        grade = ’B’;
        cout << "Very Good" << endl;
}
else if (score >= 70) {
        grade = ’C’;
        cout << "Good" << endl;
}
else if (score >= 60)
        grade = ’D’;
else
        grade = ’F’;
------------------------------------------------------------
// Ch is a char
if ((Ch >= ’a’) && (Ch <= ’z’)) {
        // code to process lower-case leter
```

```
}
else if ((Ch >= ’A’) && (Ch <= ’Z’)) {
        // code to process upper-case leter
 }
else if ((Ch >= ’0’) && (Ch <= ’9’)) {
        // code to process digit character
}
else {
        // code to process non-letter/non-digit characters
}
```

SWITCH Multiple Selection Statement

Instead of using multiple if/else statements C++ also provides a special control structure, switch.

For an integer variable x, see note, the switch(x) statement tests whether x is equal to the constant values x1, x2, x3, etc. and takes appropriate action. The default option is the action to be taken if the variable does not have any of the values listed.

Syntax:

```
switch(x )

{

        case x1 :

                statements1 ;

                break;

        case x2 :

                statements2 ;

                break;

        case x3 :

                statements3 ;

                break;

default:

                statements4 ;

                break;

}
```

The break statement causes the program to proceed to the first statement after the switch structure. Note that the switch control structure is different to the others in that braces are not required around multiple statements. If you forget to put in one of the break statements, then execution can continue from one case statement into another. Sometimes you intend this to happen, but more often than not it is not intended and a source of many bugs. Be careful to include all the break statements unless you are very sure there are some you do not need.

Note

The variable x must be either an integer type, or one that can evaluate to an integer. In practice this means you can successfully use switch on ints and chars, since single characters are represented by their so-called ASCII code and thus are on an equal footing as integers – indeed are the most basic of the integer types. Do not attempt to use switch on a variable which is a float or a double or a string. It almost certainly won't work, and even if it appears to work, it could fail at some later point.

Switch and if-else Differences:

- Switch is usually more compact than lots of nested if-else and therefore, more readable.
- If you omit the break between two switch cases, you can fall through to the next case.
- If allows complex expressions in the condition while switch wants a constant or variable.
- Switch cannot be used to test multiple conditions.
- Switch only accepts only some data types.