The Zen of C++

Chapter 3: Interactive Input, Expressions, and Formatting

Adnan Zejnilovic

Mathematical Library Functions

- #include <cmath>
  - abs(a)
  - exp(a)
  - sqrt(a)
  - pow(a,n) // a is a number and n is the exponent

Mathematical Library Functions

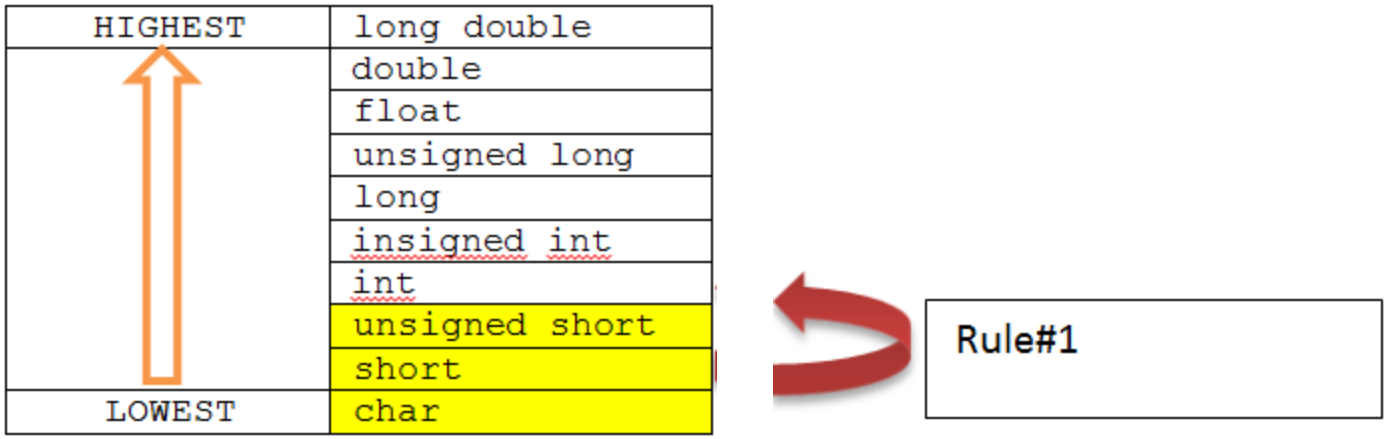
```
1   int main()
2   {
3       int number,       // to hold user  number
4           myExponent,   // to hold user exponent
5           myPower;      // to hold the result of number ^ myPower
6       double myRoot;    // to hold the result of sqrt(myRoot)
7
8       cout << "Demonstrating the pow function: " << endl;
9       cout << "Enter a number: ";
10      cin >> number;
11      cout << "What power would you like to raise " << number << " ? :";
12      cin >> myExponent;
13      myPower = pow(static_cast<double>(number), myExponent);
14      cout << number << " ^ " << myExponent
15           << " = " << myPower << endl;
16
17      cout << "\n\n\nDemonstrating the sqrt function: " << endl;
18      myRoot = sqrt(myPower);
19      cout << "sqrt(" << myPower << ") = " << myRoot << endl;
20
21      return 0;
22  }
```

Type Conversion

- C++ operator and operands
- If operands are of different data type, C++ attempts to convert them to same type
- Implicit (automatic) conversion
  - a.k.a “type coercion”
- Promotion
- Demotion

Type Conversion

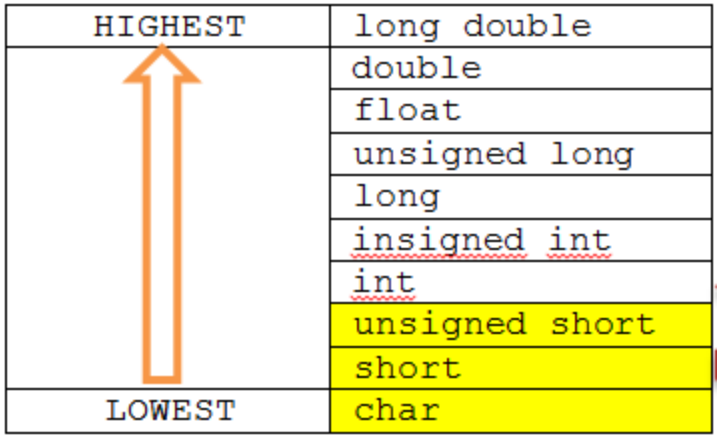
- Data type rankings:



R1: char, short, unsigned short are automatically promoted to an int

Type Conversion

- Data type rankings:



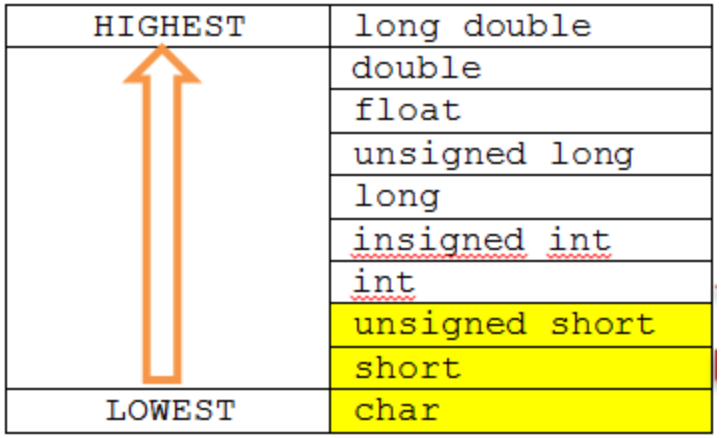

```
int x = 10;
double y = 5.43;

cout << x + y;
```

R2: when an operator works with two operands of different data types, the lower ranking operand is promoted to the data type of the high ranking operand

Type Conversion

- Data type rankings:




```
int x;
double y = 5.43;

x = y;
```

R3: when the final value of an expression is assigned to a variable, it is converted to the data type of the variable regardless of the data type rank.

Type Conversion

- R1: **char**, **short**, and **unsigned short** are automatically promoted to **int**
- R2: when an operator works with two operands of different data types, the **lower ranking opera**nd is **promoted** to the **data type** of the **high-ranking** operand.
- R3: when the **final value of an expression** is **assigned** to a **variable**, it is converted to the data type of the variable regardless of the data type rank.

Type Casting

```
double myValue = 1.23456;
```

- How would you convert this to a different data type ?
- `static_cast<DataType>(value)`

```
1    int main()
2    {
3        const int numTests = 3;
4        int test1=85, test2=100, test3=87;
5        double average;
6
7        average = (test1+test2+test3)/numTests;
8        cout << "Your average is : " << average << endl;

9        return 0;
10   }
```

Legacy Casts

```
average = (test1+test2+test3)/(double)numTests;  // C cast syntax
average = (test1+test2+test3)/double (numTests); // C++ cast syntax
```

Expressions

- Q: What is an expression?
- A: Anything that evaluates to numeric value.
- Expression vs. “Expression Statement”

Expressions

Expressions with arithmetic operators:

- 3 + 5 * 2
- 3 + (12-4) * ((14/2 -8)*3)
- cout << "3 + 5 * 2 is : " << 3 + 5 * 2 << endl;

```
int result = 3 + 5 * 2;
cout << "3 + 5 * 2 is : "
     << result
     << endl;
```

Operators

- Classification?
  - Unary
  - Binary
  - Ternary

Operators - Unary

- Negation (-)
- Increment (++)
- Decrement (--)

Operators - Increment

```
x = x + 1;
++x;
x +=1;
```

Operators - Decrement

```
x = x - 1;
--x;
x -=1;
```

Prefix vs. Postfix

Prefix:

- operation takes place first, then the operand is used

Postfix:

- Operand is first used, then the operation takes place

Binary Operators

Operator	C++ Symbol	Example
Addition	+	5 + 3
Subtraction	-	5.4 – 3
Multiplication	*	5 * 3.14
Division	/	5.0 / 3.0
Modulus	%	5 % 3

Relational Operators

Operator	C++ Symbol	Example	Means	Evaluates[2]
Equal	==	x == y	Is operand x equal to operand y?	False
Greater than	>	x > y	Is operand x greater than operand y?	True
Less than	<	x < y	Is operand x less than operand y?	False
Greater or equal	>=	x >= y	Is operand x greater or equal to operand y?	True
Less or equal	<=	x <= y	Is operand x less or equal to operand y?	False
Not equal	!=	x != y	Is operand x not equal to operand y?	False

Conditional Statements

- Execute only if certain conditions are met
- Designed to mimic real world situations
  - "if it is raining outside, take an umbrella"
- Start with the keyword "if"

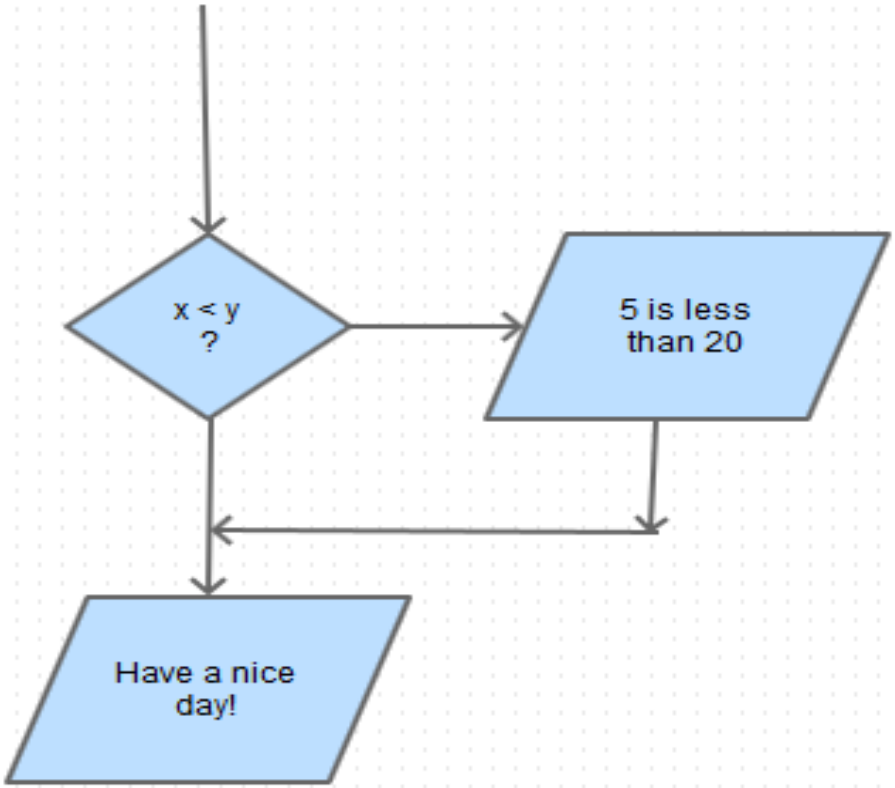
```
if (expr)
    statement1;
statement2;
statement3;
```

- Only if expr evaluates to true, then "statement1" will execute

Simple if Statement

```
1    int main()
2    {
3        int x = 25;
4        int y = 20;
5        if (x < y)
6           cout << x << " is less than " << y << endl;
7
8        cout << "Have a nice day!" << endl;
9        return 0;
10   }
```

A Flowchart for a Simple if Statement



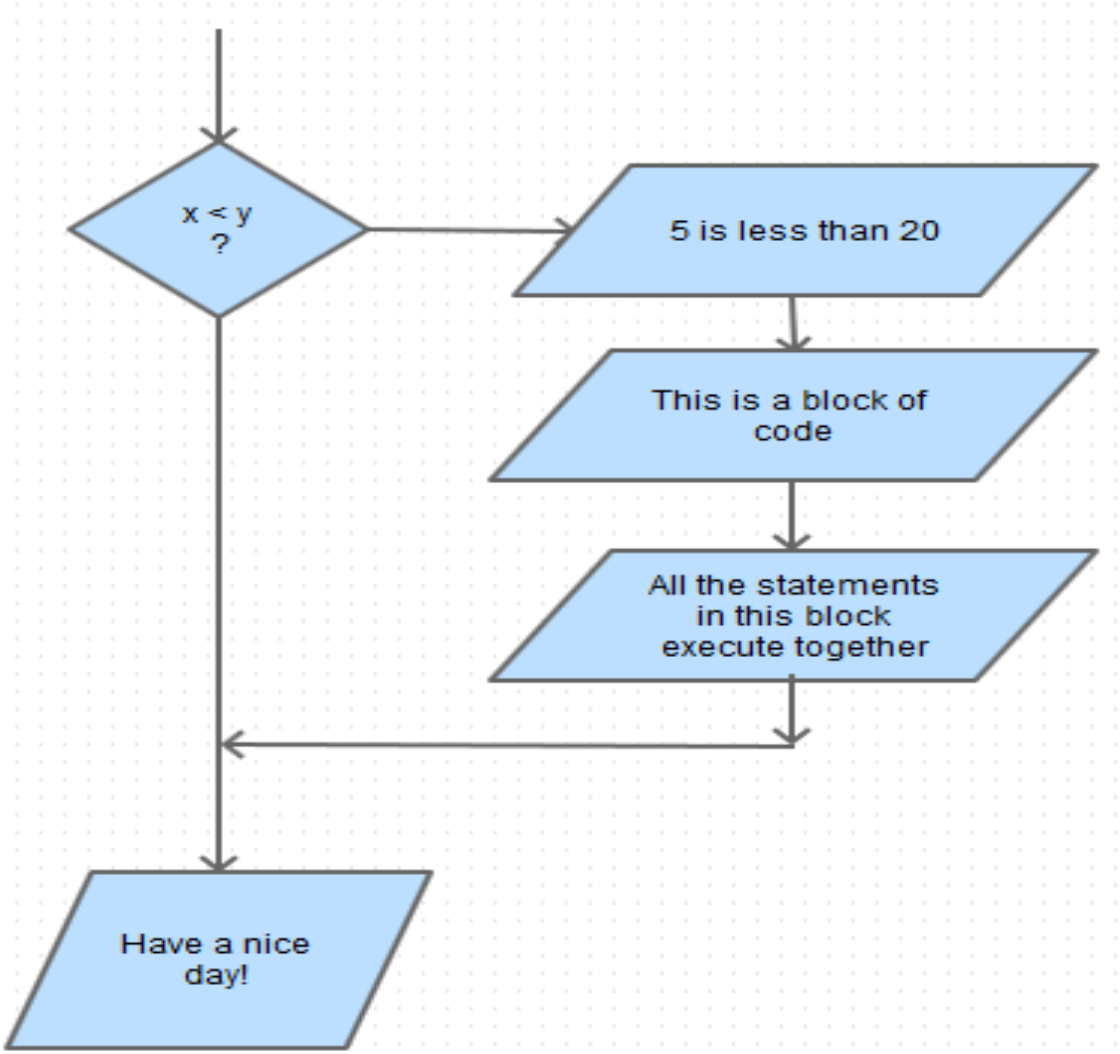
Flowcharts

- http://en.wikipedia.org/wiki/Flowchart
- https://www.programiz.com/article/flowchart-programming
- https://www.draw.io/

Block of Code

```
1   int main()
2   {
3       int x = 5;
4       int y = 20;
5       if (x < y)
6       {
7           cout << x << " is less than " << y << endl;
8           cout << "This is a block of code." << endl;
9           cout << "All the statements in this block execute together."
    << endl;
10      }
11
12      cout << "Have a nice day!" << endl;
13      return 0;
14  }
```

If Statement - Block of Code



Common Mistakes

- Using the assignment (=) operator instead of the equal (==) operator
- Semicolon at the end of if statement

```
1       int main()
2       {
3           int x = 25;
4           int y = 20;
5           if (x < y);
6           {
7               cout << x << " is less than " << y << endl;
8               cout << "This is a block of code." << endl;
9               cout << "All the statements in this block execute together." <<
        endl;
10          }
11
12          cout << "Have a nice day!" << endl;
13          return 0;
14      }
```

Float Comparisons

- Very dangerous
- What every Computer Scientist should know about float comparisons:

http://docs.oracle.com/cd/E19422-01/819-3693/ncg_goldberg.html

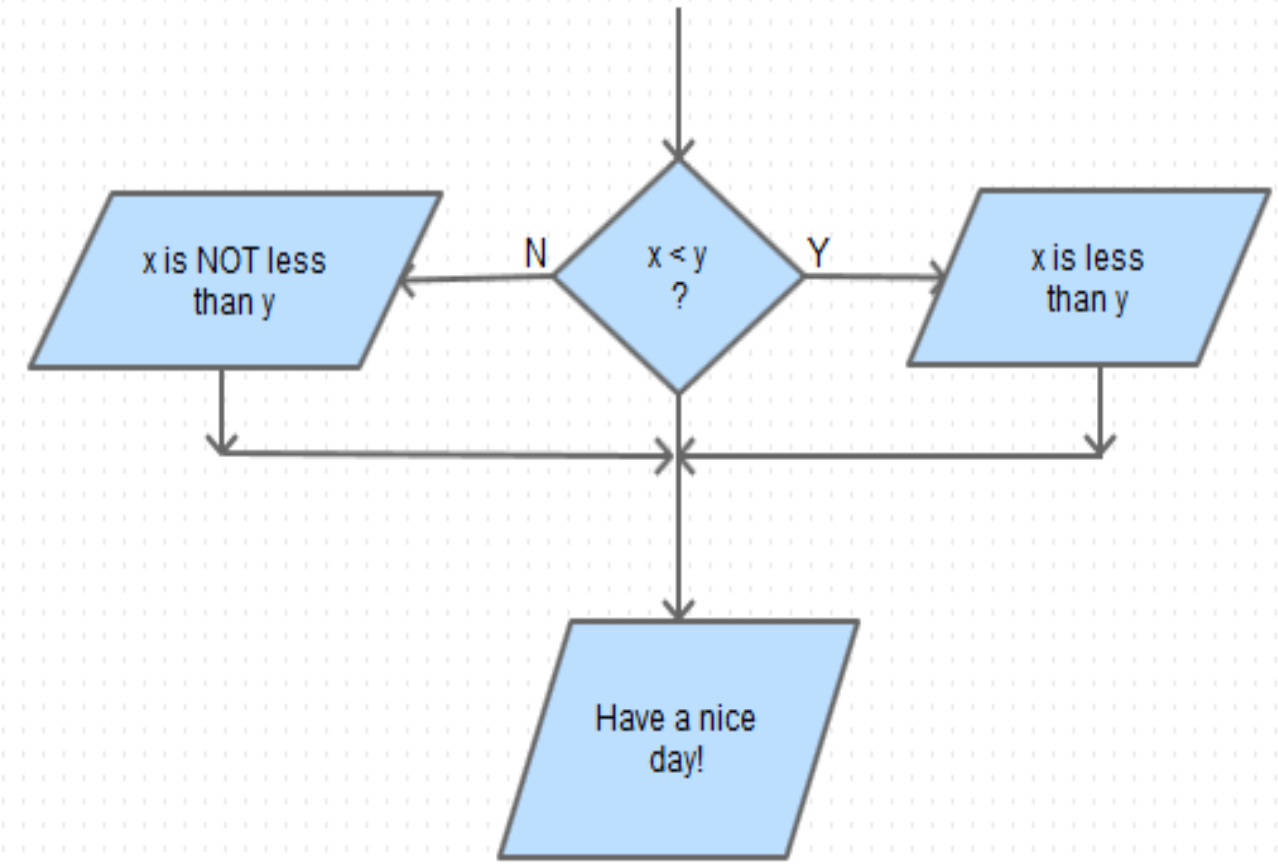
- Also this: http://www.cygnus-software.com/papers/comparingfloats/comparingfloats.htm

if-else

Single Statement	Block of Code

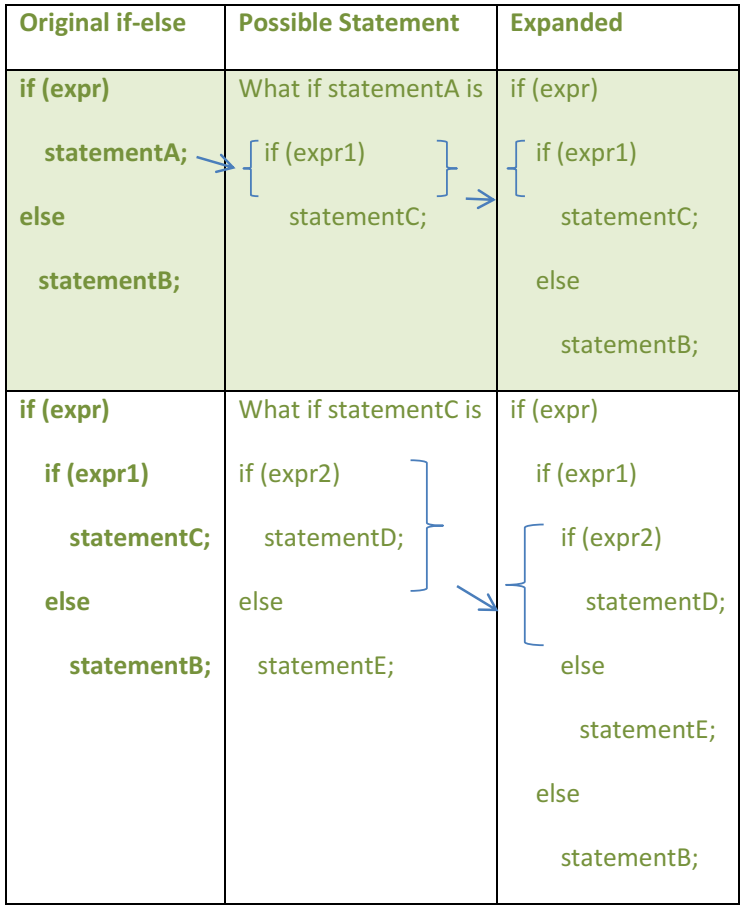
`if (expr)`
`  statementA;`
`else`
`  statementB;`
	`if (expr)`
	`{`
	`  statement(s);`
	`}`
	`else`
	`{`
	`  statement(s);`
	`}`

if else Flowchart



Nested if Statements

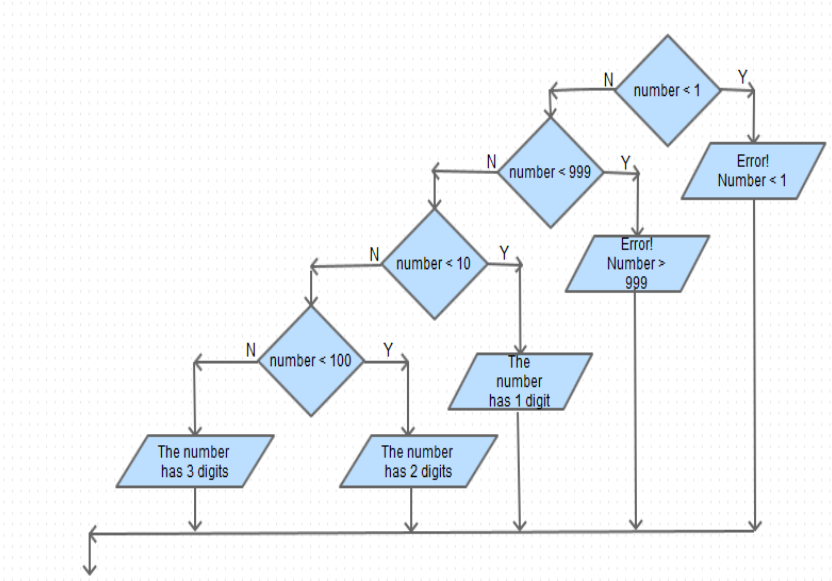
- Sometimes the if statements will not be adequate to model the logic used to solve a particular problem.
- For those situations, C++ provides alternatives such as nested if statements and if – else – if statements.
- Any statement can be substituted for an “if” or “if-else” statement



Nested if Statements (contd.)

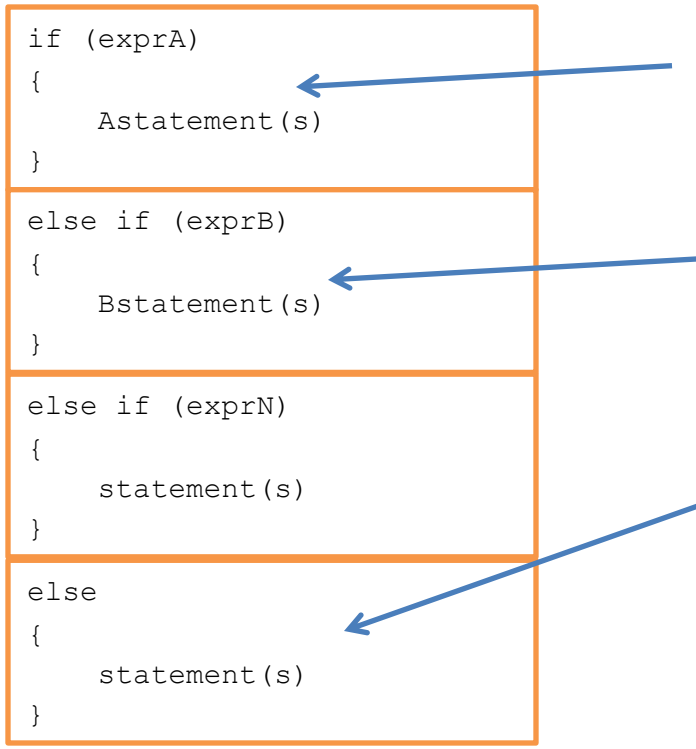
Sample program and a corresponding flowchart depicting nested if statements

```
int main()
{
    int number;
    cout << "Enter a number between 1 and 999: ";
    cin >> number;
    if (number < 1 )
    {
        cout << "ERROR! The number is less than 1! " << endl;
    }
    else
    {
        if (number > 999)
        {
            cout << "ERROR! The number is greater than 999! " << endl;
        }
        else
        {
            if (number < 10)
               cout << "The number has ONE digit." << endl;
            else
            {
                if (number < 100)
                   cout << "The number has TWO digits. " << endl;
                else
                   cout << "The number has THREE digits. " << endl;
            }
        }
    }
    return 0;
}
```



The if else if Statements

- Nested if statements could become complex resulting in difficult to understand source code.
- Fortunately, there is an easier way to implement the same logic through the if–else –if statements.



- The first test is exprA. If it evaluates to true, only the block of code "Astatement(s)" associated with that if statements will execute.
- If not, the program attempts the exprB. If the expression evaluates to true, then only the block of code associated with this (else if) statement will execute.
- If no expressions evaluate to true, the trailing else statement, if present will execute.
- Most of the time, it will be possible to rewrite nested if statements as if-else-if statements resulting in more readable code, which is easier to debug and maintain.

Nested if Statements Rewritten as if else if

Code with Nested if Statements

```
int main()
{
    int number;
    cout << "Enter a number between 1 and 999: ";
    cin >> number;
    if (number < 1 )
    {
        cout << "ERROR! The number is less than 1! " << endl;
    }
    else
    {
        if (number > 999)
        {
            cout << "ERROR! The number is greater than 999! " << endl;
        }
        else
        {
            if (number < 10)
               cout << "The number has ONE digit." << endl;
            else
            {
                if (number < 100)
                   cout << "The number has TWO digits. " << endl;
                else
                   cout << "The number has THREE digits. " << endl;
            }
        }
    }
    return 0;
}
```

Code with if else if Statements

```
int main()
{
    int number;
    cout << "Enter a number between 1 and 999: ";
    cin >> number;
    if (number < 1 )
    {
        cout << "ERROR! The number is less than 1! " << endl;
    }
    else
    {
        if (number > 999)
        {
            cout << "ERROR! The number is greater than 999! " << endl;
        }
        else
        {
            if (number < 10)
               cout << "The number has ONE digit." << endl;
            else
            {
                if (number < 100)
                   cout << "The number has TWO digits. " << endl;
                else
                   cout << "The number has THREE digits. " << endl;
            }
        }
    }
    return 0;
}
```

The switch Statement

- The switch statement is very similar to the if else if statement.
- The only difference is that the switch statement works only with integer (char) data types.
- Only the code in ***one*** of the case statements *will execute*.
- In case there is no default, and there is no match, none of the case statements will execute.
- Excellent choice for processing menu choices

Switch Statement Syntax:

```
switch(expr)
{
    case value1:
        statement(s);
        break;
    case value2:
        statement(s);
        break;
    . . . . . .
    . . . . . .
    case valueN:
        statement(s);
        break;
    default:
        statement(s);
}
```

The switch Statement (contd)

- The ***break*** statement is responsible for terminating the case and *transferring* the program control to the statement following the switch statement

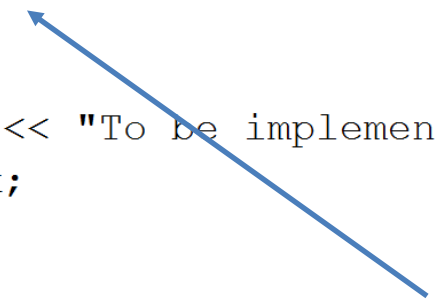
Switch Statement Syntax:

```
switch(expr)
{
   case value1:
      statement(s);
      break;
   case value2:
      statement(s);
      break;
   . . . . . .
   . . . . . .
   case valueN:
      statement(s);
      break;
   default:
      statement(s);
}
Statement after the switch
```

The switch Statement (contd)

If the options for some case statements are the same then it is possible to stack these case statements by omitting the break statement:

```
switch(number)
{
    case 1:
        cout << "Check Account Balance" << endl;
        break;
    case 2:
    case 3:
    case 4:
        cout << "To be implemented later" << endl;
        break;
}
```



case 2, 3, and 4 are the same
because there is no break statement between them

The switch Statement (contd)

These two statements are equivalent

```
switch(number)
{
case 1:
    cout << "Check Account Balance" << endl;
    break;
case 2:
    cout << "Withdraw" << endl;
    break;
case 3:
    cout << "Deposit" << endl;
    break;

case 4:
    cout << "Thank you for using my ATM" << endl;
    cout << "Exiting ..." << endl;
    break;
default:
    cout << "Please enter 1, 2, 3, or 4" << endl;
}
```

```
if (number == 1)
    cout << "Check Account Balance" << endl;
else if (number == 2)
    cout << "Withdraw" << endl;
else if (number == 3)
    cout << "Deposit" << endl;
else if (number == 4)
{
    cout << "Thank you for using my ATM" << endl;
    cout << "Exiting ..." << endl;
}
else
    cout << "Please enter 1, 2, 3, or 4" << endl;
```

Logical Operators

- Logical operators provide a mechanism to combine two or more relational expressions into a single expression that evaluates either to true or false.
- Using logical operators it is possible to create more complex conditions by combining two or more simple expressions.

Operator	C++ Symbol	Example
AND	&&	(expressionA && expressionB)
OR	||	(expressionA || expressionB)
NOT	!	!expressionA

Logical AND (&&) Operator

- A binary operator used to combine the two or more relational expressions into one expression.
- The operands are expressions
- The logical AND operator requires both of its operands to be true in order to evaluates the expression is true.
- If any of the operands is false to false then the logical AND operator will evaluate the expression to false.
- Lazy-evaluation – evaluates the first operand and if it evaluates to false, the whole expression is evaluated to false

Logical AND Truth Table

Operand1	Operand2	Operand1 && Operand2
T	T	T
T	F	F
F	T	F
F	F	F

Example:

```
cout << "Please enter a number between 1 and 100: ";
cin >> number;

if (number >=1 && number <=100)
    cout << "Correct Range" << endl;
else
    cout << "Incorrect range!" << endl;
```

Logical OR (||) Operator

- A binary operator used to combine the two or more relational expressions into one expression.
- The operands are expressions
- The logical OR operator requires ***only one*** of the operands to be ***true*** in order for it to evaluates the entire expression as ***true***.
- It will evaluate the expression to false if both of the operands are false.

Logical OR Truth Table

Operand1	Operand2	Operand1 || Operand2
T	T	T
T	F	T
F	T	T
F	F	F

Example:

```
cout << "Please enter a number between 1 and 100: ";
cin >> number;

if (number < 1 || number >100)
     cout << "The number is outside the allowed range" << endl;
else
     cout << "Correct range!" << endl;
```

Logical NOT (!) Operator

- A ***unary*** operator used to reverse the truth of an expression
- If an expression evaluates to true, placing a logical NOT operator in front of it will result in it evaluating to false (and vice versa)

Logical NOT Truth Table

Operand1	!Operand1
T	F
F	T

Example:

```
int x = 10;
int y = 20;

if (!(x < y))
    cout << x << " is less than " << y << endl;
else
    cout << x << " is greater than " << y << endl;
```

```
10 is greater than 20

Process returned 0 (0x0)   execution time : 0.033 s
Press any key to continue.
```

Logical Operator Precedence

- There is a hierarchy of execution among logical operators.
- The logical NOT operator has the highest precedence, followed by the logical AND, followed by logical OR.
- The following table displays operator precedence hierarchy:

Precedence	Operator					
1	!	-	++	--		
2	*	/	%			
3	+	-				
4	<	<=	>	>=		
5	==	!=				
6	&&					
7	||					
8	=	+=	-=	*=	/=	

Formatting Output

- The cout object has limited way of formatting data.
- Basic formatting manipulators defined in iostream header
- To format output (and input) in a more elaborate way you need to include iomanip header which contains numerous manipulators

Formatting Output

manipulator	Header	description
endl	iostream	Insert newline character
left	iostream	Left justify the output
right	iostream	Right justify the output (default)
fixed	iostream	Display floating-point values as decimal
scientific	iostream	Display floating-point values in scientific notation
showpoint	iostream	Forces the decimal point to be printed
noshowpoint	iostream	Reverses showpoint
setfill (ch)	Iomanip	Specify some other character (ch) than a space (default) to pad the output
setw (w)	iomanip	Specifies the minimum number os spaces (w) for the next numeric or string value
setprecision (n)	iomanip	Set floating-point precision to n digits

Example: setprecission

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    double number = 1.234567;
    cout << setprecision(6) << number << endl;
    cout << setprecision(5) << number << endl;
    cout << setprecision(4) << number << endl;
    cout << setprecision(3) << number << endl;
    cout << setprecision(2) << number << endl;
    cout << setprecision(1) << number << endl;
    return 0;
}
```

```
1.23457
1.2346
1.235
1.23
1.2
1

Process returned 0 (0x0)   execution time : 0.036 s
Press any key to continue.
```