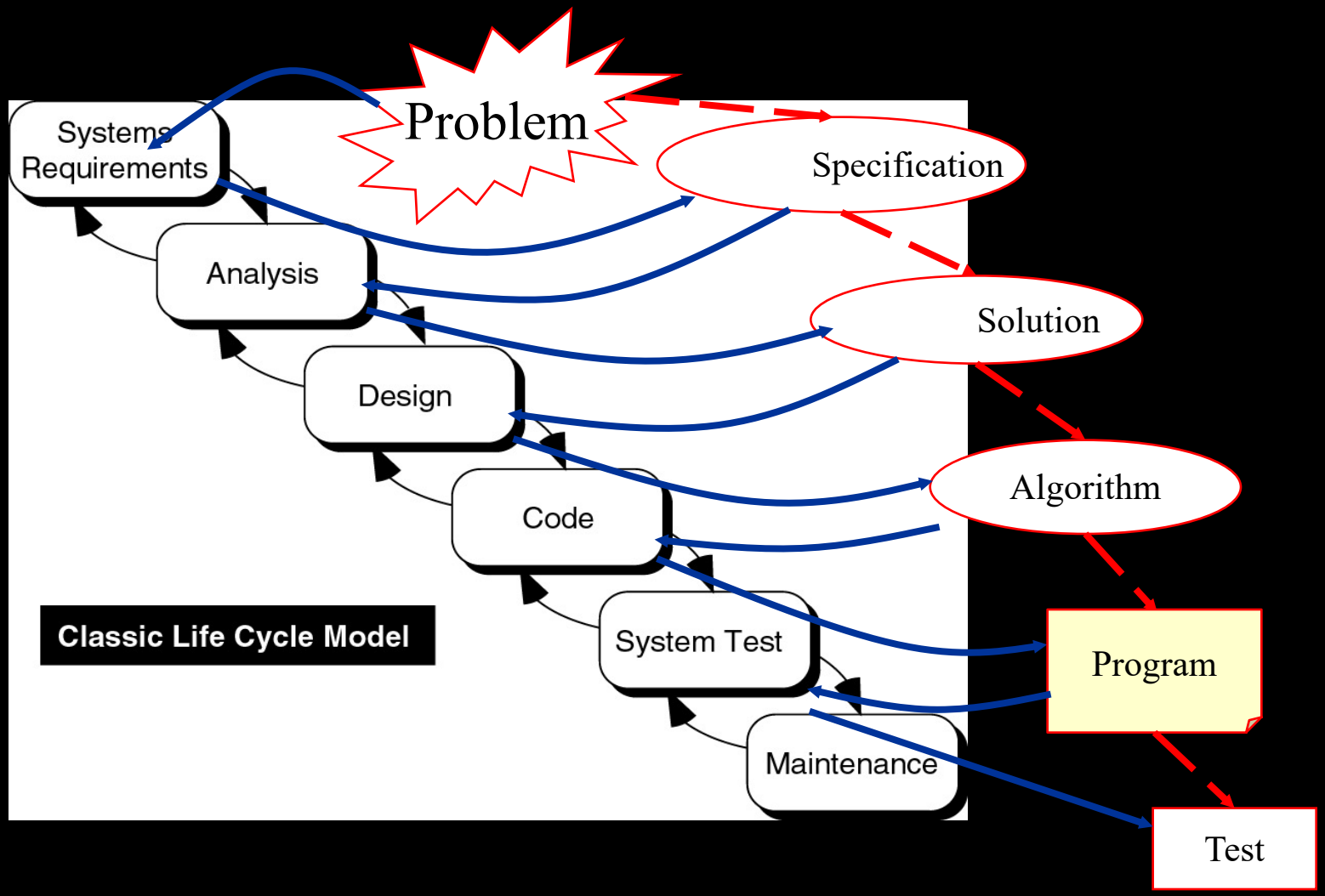
Selection Structures

A Programming Example

- The Problem:
  Write a program that takes as input a given length expressed in feet and inches. It then converts and outputs the length in centimetres.

System Development Life Cycle



The Problem Specification

- Input: Length in feet and inches
- The program will need to find the equivalent length in centimetres
  One inch is equal to 2.54 centimetres
- Output: Equivalent length in centimetres

Analysing the Problem

- The first thing the program needs to do is convert the length given in feet and inches to inches only.

Next, use the conversion formula

1 inch = 2.54 centimetres

to find the equivalent length in centimetres.

Analysing the Problem

- To convert the length from feet and inches to inches, multiply the number of feet by 12, as 1 foot is equal to 12 inches, and add the given inches.

Then apply the conversion formula, that is, 1 inch = 2.54 centimetres, to find the length in centimetres.

A General Solution

-
  1. get the length in feet and inches
  2. convert the length into total inches
  3. convert total inches into centimetres
  4. output centimetres

Program Design

- The algorithm:

1. Prompt the user for the input
2. Get the data
3. Find the length in inches
4. Output the length in inches
5. Convert the length to centimetres
6. Output the length in centimetres

Program Implementation

- The program will begin with comments that document its purpose and functionality.
- The program will use input statements to get data into the program and output statements to print the results.

Program Implementation

- The data will be entered from the keyboard and the output will be displayed on the screen, so the program must include the header file `iostream`.
- The first statement of the program, after the comments as described above, will be the preprocessor directive to include this header file.

Program Implementation

- Variables

```
int feet;           // holds given feet
int inches;         // holds given inches
int totalInches;    // holds total inches
double cm;          // holds length in cm
```

- Named Constants

```
const double cmPerInch = 2.54;
const int inchesPerFoot = 12;
```

Program Implementation

- The body of the function main has the following form

```
int main ()
{
    declare variables
    statements
    return 0;
}
```

Writing the Complete Program

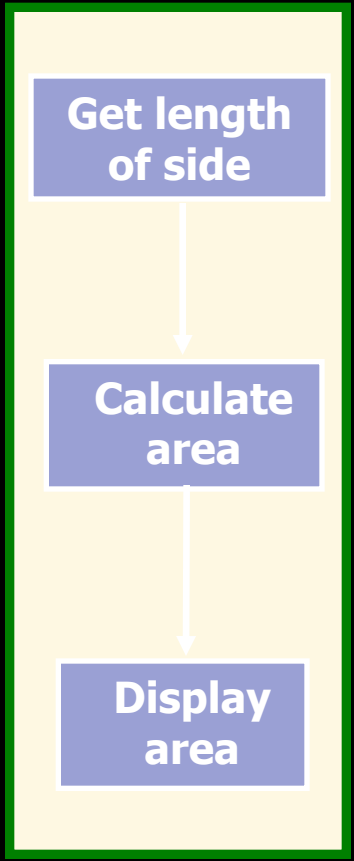
- Begin the program with comments for documentation.
- Include header files, if any are used in the program.
- Declare named constants, if any.
- Write the definition of the main.

Writing the Complete Program

```
// Program Description: Converts a given length
// from feet and inches to cm

#include <iostream>
using namespace std;

const double cmPerInch = 2.54;
const int inchesPerFoot = 12;

int main()
{
    int feet, inches, totalInches;
    double cm;
    cout << "Please enter length in feet and inches: ";
    cin >> feet >> inches;
    totalInches = inches + feet*inchesPerFoot;
    cout << "\nThe length in inches is: " << totalInches << endl;
    cm = totalInches * cmPerInch;
    cout << "The length in centimetres is: " << cm << endl;

    return 0;
}
```

Flowchart of a Simple Program

```
//Calculates the surface area of a cube
#include <iostream>
using namespace std;

int main()
{
    int area, length;
    cout << "Enter length of side in cm:";
    cin >> length;
    area = length * length * 6;
    cout << "Area is " << area << " sq cm\n";
    return 0;
}
```

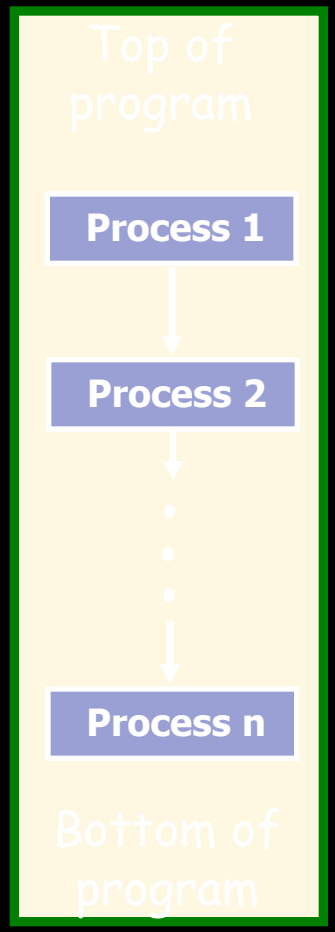


Program Flow of Execution

The statements in the previous program are executed in sequence.

The statements are executed one after the other from the top to the bottom of the program.

This is called consecutive or sequential code execution.



Program Flow of Execution

- The three main ways the execution of the statements in a program may progress are:
  - Consecutive or sequential execution
  - Choice (or selection)
  - Repetition (or iteration)

Program Flow Control Constructs

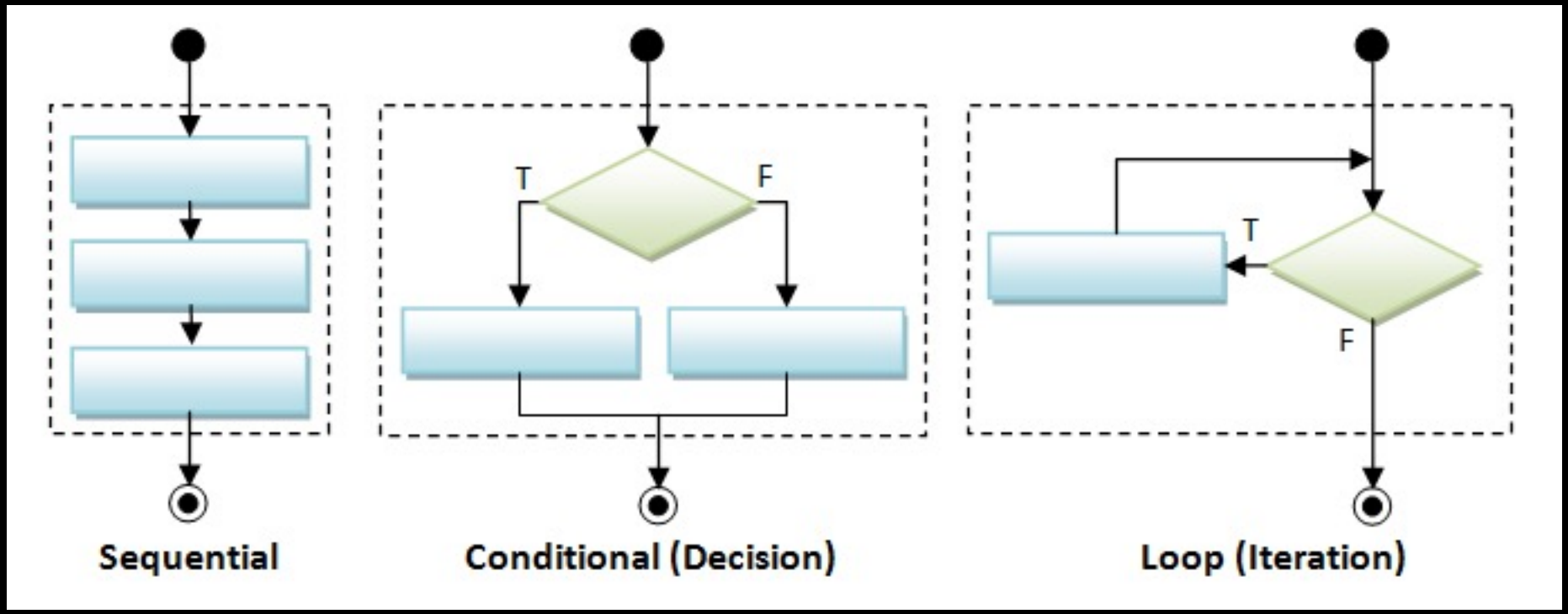
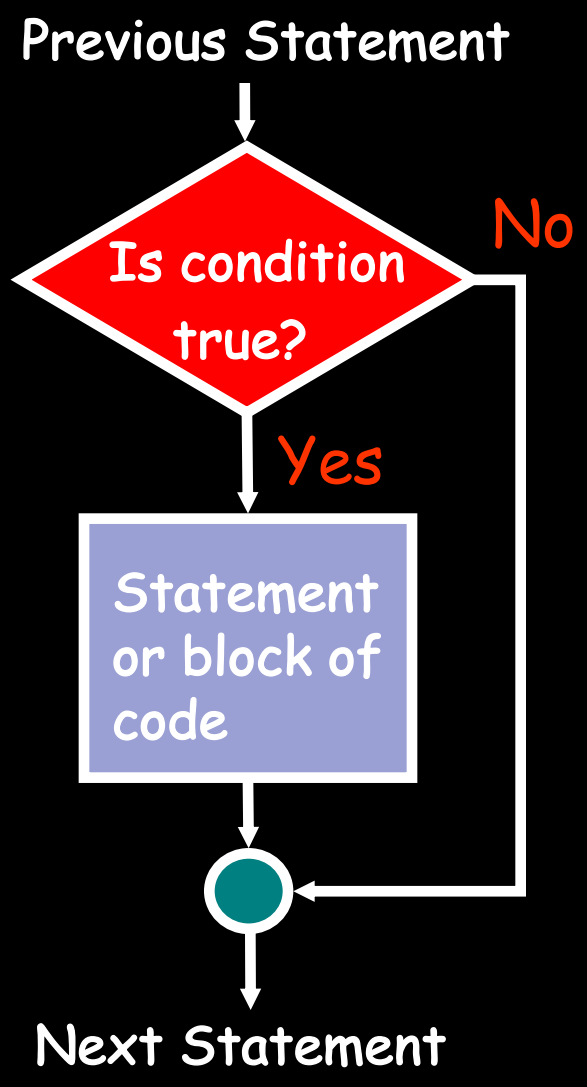


Diagram from https://www3.ntu.edu.sg/home/ehchua/programming/cpp/cp1_Basics.html

The Conditional Statement Flowchart



The Conditional Statement Pseudocode

```
BEGIN
    ...
    IF condition THEN
        statement
    ENDIF
    ...
END
```

executed if condition is true

- Determines whether statement (or block of code) is executed by evaluating the condition.
- If the condition is true (non-zero), statement (or block of code) is executed.

Like when we use if in current language

condition

```
IF it's raining THEN
    I'll take my umbrella
ENDIF
```

executed if condition is true

If it is raining the condition is true and as a consequence `I'll take my umbrella`

The Conditional Statement in C++

```
if (logical expression)
    statement
```

- As with any C++ statement the whitespace is not required, so

```
if (logical expression) statement
```

- is also valid.
- Note there is no ; after the)

The condition in an if Statement

- The logical expression can be created using

  relational expressions, logical operations or a combination of these

  and

  may also contain arithmetic expressions

  Example:

```
if (!(a*b > a/3) && (a <= 9))
    ...
```

Relational Operators

Operator	Meaning	Example
<	less than	`3 < 10`
<=	less than or equal to	`width <= 5`
>	greater than	`degrees > 43.5`
>=	greater than or equal to	`age >= 5`
==	equal to	`mark == 85`
!=	not equal to	`distance != 12.3`

All six relational operators are binary

Relational Expressions

Relational expressions have logical values:

false (e.g. 3 == 5) or,

true (e.g. 3 < 5)

If age=7, the value of age >= 5 is true

If age=2 , the value of age >= 5 is false

Although a boolean value may be stored as 0 or 1, do not think of these values as anything other than true or false.

Relational Operators Precedence

Operator	Precedence
< <= > >=	higher
== !=	lower

An example

What is the output of this program?

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

int main()
{
    int a =  5, b = -3;

    cout << a << " <  " << b
         << " is  " << (a < b) << endl;
    cout << a << " == " << b
        << " is  " << (a == b) << endl;
    cout << a << " != " << b
        << " is  " << (a != b) << endl;
    cout << a << " >= " << b
        << " is  " << (a >= b) << endl;

    return 0;
}
```

Answer:

```
5 <  -3 is  1
5 == -3 is  0
5 != -3 is  1
5 >= -3 is  1
```

Wait …. Wait …. Wait

- Didn't we just say not to consider boolean values as 0 or 1?
- Unfortunately C++ cannot print out (or enter) boolean values.
- So, avoid doing so in programs.
- Use if tests on both input and output.

Wait …. Wait …. Wait

```
int a =  5, b = -3;
bool comp;

comp = (a>b);

if (comp)
    cout << "true\n";
```

Wait .... Wait .... Wait

```
bool under20=false;
char ch;

cout << "Are you aged under 20? ";
cin << ch;
if (ch == 'Y')
    under20 = true;
```

But what if a lower case y were entered?

We would need to check for that as well.

Logical Operators

- Logical values can also be combined using logical operators.

- One of these has already been encountered:

  not with symbol !

  complements the meaning

  `!true` is `false`, `!false` is `true`

  `!(age<23)` is the same as `age>=23`

Logical Operators

- The other two operators are binary

  logical and with the symbol &&

  logical or with the symbol ||

  Such operations are best explained using a truth table

- ! has higher priority

Logical Operators

And – the value of **`expr1 && expr2`**

		expr2	
		true	false
expr1	true	true	false
	false	false	false

Logical Operators

Or - the value of `expr1 || expr2`

expr1 \ expr2	true	false
true	true	true
false	true	false

Logical Operators

- Note that

  `!(expr1 || expr2)` is the same as
  `!expr1 && !expr2`

  `!(expr1 && expr2)` is the same as
  `(!expr1) || (!expr2)`

Operator Precedence

- Highest
- `!` `+` `-` logical not, unary +, unary –
- `*` `/` `%` multiplication, division, mod
- `+` `-` addition, subtraction
- `<` `<=` `>=` `>` relational inequality
- `==` `!=` equal, not equal
- `&&` logical and
- `||` logical or
- `=` assignment
- lowest

Examples

- Given the following values for boolean variables **X**, **Y**, and **Z**, evaluate the logical expressions and answer T if the result is True and F if the result is False.

- `X = true, Y = false, Z = false`
-
- `!(Y || Z) || X`
- `Z && X && Y`
- `!Y || (Z || !X)`
- `Z || (X && (Y || Z) )`
- `5 || !X && Z`

Answer:

T

F

T

F

T

Example

- Write a C++ program that reads a number from standard input and prints it if the number is even.

Example

Reads in a number and prints it out if even:

```
BEGIN
 output "Enter an integer:"
 read in number
 IF number is even THEN
   printout the number
 ENDIF
END
```

```
// Reads in a number and prints it
// out if even
#include <iostream>
using namespace std;

int main()
{



   return 0;
}
```

Example

Reads in a number and prints it out if even:

```
BEGIN
 output "Enter an integer:"
 read in number
 IF number is even THEN
   printout the number
 ENDIF
END
```

```
// Reads in a number and prints it
// out if even
#include <iostream>
using namespace std;

int main()
{
   int number;

   cout << "Enter an integer: ";
   cin >> number;




   return 0;
}
```

Example

Reads in a number and prints it out if even:

```
BEGIN
 output "Enter an integer:"
 read in number
 IF number is even THEN
   printout the number
 ENDIF
END
```

```
// Reads in a number and prints it
// out if even
#include <iostream>
using namespace std;

int main()
{
   int number;

   cout << "Enter an integer: ";
   cin >> number;

   if (number % 2 == 0)
    cout << numbe << endl;
```

LOOK no semicolon here!

Example

What happens if by mistake I put a ; after (number % 2 == 0)?

```
// Reads in a number and prints it
// out if even
#include <iostream>
using namespace std;

int main()
{
   int number;

   cout << "Enter an integer: ";
   cin >> number;

   if (number % 2 == 0);
    cout << number << endl;

   return 0;
}
```

What is the problem?

- `if (number % 2 == 0);`
- is equivalent to:
- `if (number % 2 == 0)`
- `;  //empty statement`

- So the statement following ';' which is
- `cout << number << endl;`
- is always executed

Example

LOOK it's ==
not =

```
// Reads in a number and prints it
// out if even
#include <iostream>
using namespace std;

int main()
{
   int number;

   cout << "Enter an integer: ";
   cin >> number;

   if (number % 2 == 0)
        cout << number << endl;

   return 0;
}
```

Example

By mistake x=5 was typed in instead of x==5, what is the output if I enter 5?

```
// A common mistake
// should be x==5 instead of x=5
#include <iostream>
using namespace std;

int main()
{
    int x;
    cin >> x;

    if (x = 5)
        cout << "x is 5";

    return 0;
}
```

Answer: `x is 5`

Example

- Modify the example program to output the following extra line before the line printing out the even number: I'm not odd

Example

This is a compound statement - several statements inside a {} pair.

No need for following ;

Reads in a number and prints it out if even

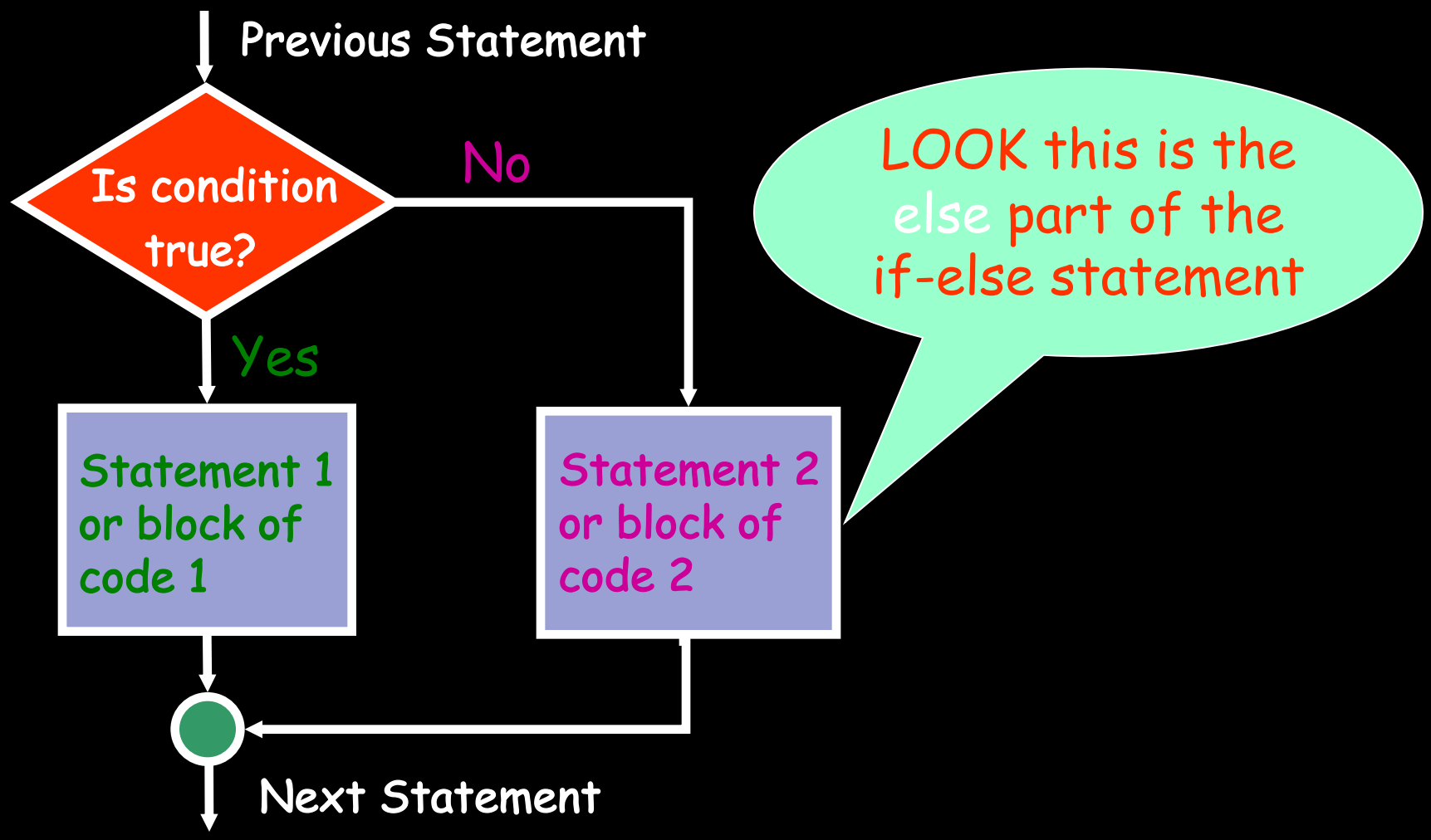
```
BEGIN
 output "Enter an integer:"
 read in number
 IF number is even THEN
   printout "I'm not odd!!"
   printout the number
 ENDIF
END
```

```
// Reads in a number and prints it
// out if even
#include <iostream>
using namespace std;

int main()
{
   int number;

   cout << "Enter an integer: ";
   cin >> number;

   if (number % 2 == 0)
   {
     cout << "I'm not odd!!";
   cout << number << endl;
   }

   return 0;
}
```

The if-else Statement Pseudocode

```
IF condition THEN
   statement 1 executed if condition is true
ELSE
   statement 2 executed if condition is false
ENDIF
```

The if-else Statement Flowchart



Notes on else

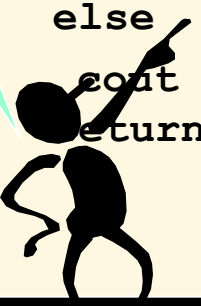
- **`else`** can only occur after an **`if`** statement
- **`else`** is only executed when the **`if`**'s statement or block does not execute
- the whole if-else structure is considered one statement

Example

Make no mistake
no semicolon here!

```
Reads in a number and f
if it is even or odd

BEGIN
 output "Enter an inte      "
 read in number
 IF number is even THEN
  printout number is even
 ELSE
  printout number is odd
 ENDIF
END
```

```
// Reads in a number and finds
   f the number is even or odd

#include <iostream>
using namespace std;

int main()
{
   int num;

   cout << "Enter an integer: ";
   cin >> num;

   if (num % 2 == 0)
    cout << num << " is even\n";
   else
    cout << num << " is odd\n";
   return 0;
}
```



Example

We can swap these two statements

if we use the condition:

`num % 2 != 0`

instead of:

`num % 2 == 0`

```
// Reads in a number and finds
// if the number is even or odd

#include <iostream>
using namespace std;

int main()
{
   int num;

   cout << "Enter an integer: ";
   cin >> num;

   if (num % 2 == 0)
    cout << num << " is even\n";
   else
    cout << num << " is odd\n";
   return 0;
}
```

Example

Reversed meaning of the condition

```
// Reads in a number and finds
// if the number is even or odd

#include <iostream>
using namespace std;

int main()
{
   int num;

   cout << "Enter an integer: ";
   cin >> num;

   if (num % 2 != 0)
    cout << num << " is odd\n";
   else
    cout << num << " is even\n";
   return 0;
}
```

More Complements

- `>` is the complement of `<=`
- `<` is the complement of `>=`
- `!expr` is the complement of `expr`

Cascaded and Nested if Statements

- if and if-else statements can contain simple or compound statements.

- Any valid C++ statement can be used including if and if-else statements.

- if and if-else statements can be included within the if and/or else part of other
- if/if-else statements.

Cascaded if Statements

```
• if (expression 1)
•     statement 1;
• else
•     if (expression 2)
•         statement 2;
•     else
•         statement 3;
```

NOTE: else statements are always paired with the closest unpaired if statement

This else is paired with this if

Cascaded if Statements

- As white space is ignored, cascaded if statements are generally implemented as follows:

```
if (expression 1)
    statement 1;
else if (expression 2)
    statement 2;
else
    statement 3;
```

"sieve"
or
"filter"

Example

```
• // Converts score to grade
• /* This program reads a test
•    score, calculates the letter grade
•    based on the absolute scale, and
•    prints it.*/

• #include <iostream>
• using namespace std;

• int main (void)
• {
      int score;
      char grade;
```

Example

```
cout << "\nEnter the test score (0-100): ";
cin  >> score;

if (score >= 90)
    grade = 'A';
else if (score >= 80)
    grade = 'B';
else if (score >= 70)
    grade = 'C';
else if (score >= 60)
    grade = 'D';
else
    grade = 'F';
```

Example

```
    cout << "The grade is: "
         << grade << endl;

    return 0;
}
```

Try this one

What is the output if

letter is equal to 'b'

letter is equal to 'z'

letter is equal to 'A'

letter is equal to 'X'

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

int main()
{
    char letter;

    cout << "Enter a character: ";
    cin >> letter;
    if (letter >= 'a')
        cout << "S1";
    else if (letter <= 'z')
        cout << "S2";
    else if (letter >= 'A')
        cout << "S3";
    else if (letter <= 'Z')
        cout << "S4";
    return 0;
 }
```

Nested if Statements

statement 1:

```
if (distance > 1000)
    cout << "long distance ";
```

statement 2:

```
if (time < 2)
  statement 1
else
   cout << "not quick";
```

Nested if Statements

- Is this statement 2?

Answer: NO

This else is paired with this if

```
if (time < 2)
   if (distance > 1000)
      cout << "long distance ";
else
   cout << "not quick";
```

Nested if Statements

- **This is statement 2:**

```
if (time < 2)
{
     if (distance > 1000)
          cout << "long distance";
}
else
     cout << "not quick";
```

The Dangling else Problem

- This example illustrated a common error known as the "dangling else".
- This problem occurs when there is no matching else for every if.
- Remember that the C++ compiler always pairs an else to the most recent if in the current block.

Conditional Expressions

- Conditional expressions provide a alternative way of expressing simple
- if-else statements

- Conditional expressions use the conditional operator ? :

Conditional Expressions

- The syntax of a conditional expression is:

condition

- `expression1 ? expression2 : expression3`

executed if condition is true

executed if condition is false

- Note: the conditional operator is a ternary operator (i.e. it takes 3 operands)

Conditional Expressions

- How do we interpret this statement?
- `cout << (expression1 ? expression2 : expression3);`
- If the value of **`expression1`** is non-zero (true) **`expression2`** is printed; otherwise, **`expression3`** is printed
- The statement either prints the value of **`expression2`** or of **`expression3`**

Conditional Expressions

- The meaning of a conditional expression is lost in a jumble of operators, without easily being able to follow the logic of the if/else process.

- **DO NOT USE**

The `switch` Statement

- We have seen that cascaded if statements are used when one instruction (or set of instructions) must be selected from many possible alternatives

the sieve concept

- The switch statement provides a convenient alternative for cases where the value of an integer expression is compared to a specific value (not a range of values)

The switch Statement

- The syntax of a switch statement:

```
switch (expression)
{
    case const_expr1:
        statement1;
        break;
    case const_expr2:
        statement2;
        break;
    case const_expr3:
        statement3;
        break;
    default
        statement
}
```

Expression's value must be integer

A constant expression

Causes exit from switch statement

Execution only gets here if none of the constant expressions equals the value of the expression

LOOK no break here!

```
    cas
            ut << "Good Day\n";
            ut << "Odds have it!\n";
            eak;
    case 
    case 4
            c t << "Good Day\n";
            co t << "Evens have it!\n";
            bre k;
    default:
            cout << "Good Day, I'm confused!\n";
              cout << "Bye!\n";
}
```

```
switch (printFlag)
{
    case 1:
    case 3:
        cout << "Good Day\n";
        cout << "Odds have it!\
        break;
    case 2:
    case 4:
        cout << "Good Day\n";
        cout << "Evens have it!
        break;
    default:
        cout << "Good Day, I'm confused!\n";
          cout << "Bye!\n";
}
```

The break statement causes an immediate exit from the switch statement

```
switch (printFlag)
{
    case 1:
    case 3:
        cout << "Good Day\n";
        cout << "Odds have it!\n";
        break;
    case 2:
    case 4:
        cout << "Good Day\n";
        cout << "Evens have it!\n";
        break;
    default:
        cout << "Good Day, I'm confused!\n";
          cout << "Bye!\n";
}
```

What is the output of this switch if `printFlag=1`?

Answer:

```
Good Day
Odds have it!
```

```
switch (printFlag)
{
    case 1:
    case 3:
        cout << "Good Day\n";
        cout << "Odds have it!\n"
        break;
    case 2:
    case 4:
        cout << "Good Day\n";
        cout << "Evens have it!\n";
        break;
    default:
        cout << "Good Day, I'm confused!\n";
          cout << "Bye!\n";
}
```

What is the output of this switch if `printFlag=2`?

Answer:

```
Good Day
Evens have it!
```

```
switch (printFlag)
{
    case 1:
    case 3:
        cout << "Good Day\n";
        cout << "Odds have it!\n";
        break;
    case 2:
    case 4:
        cout << "Good Day\n";
        cout << "Evens have it!\n";
        break;
    default:
        cout << "Good Day, I'm confused!\n";
          cout << "Bye!\n";
}
```

What is the output of this switch if `printFlag=3`?

Answer:

```
Good Day
Odds have it!
```

```
switch (printFlag)
{
    case 1:
    case 3:
        cout << "Good Day\n";
        cout << "Odds have it!\n"
        break;
    case 2:
    case 4:
        cout << "Good Day\n";
        cout << "Evens have it!\n";
        break;
    default:
        cout << "Good Day, I'm confused!\n";
          cout << "Bye!\n";
}
```

What is the output of this switch if `printFlag=4`?

Answer:

```
Good Day
Evens have it!
```

```
switch (printFlag)
{
    case 1:
    case 3:
        cout << "Good Day\n";
        cout << "Odds ha
        break;
    case 2:
    case 4:
        cout << "Good Day\n";
        cout << "Evens have it!\n";
        break;
    default:
        cout << "Good Day, I'm confused!\n";
          cout << "Bye!\n";
}
```

What is the output of this switch if `printFlag=5`?

Answer:

```
Good Day, I'm confused!
Bye!
```

`switch` structure summary

- `switch`, `case`, `break`, and `default` are reserved words.
- In a switch structure, the expression is evaluated first.
- The value of the expression is used to determine which case statement is selected for execution.

`switch` structure summary

- One or more statements may follow a case label.
- Braces are not needed to turn multiple statements into a single compound case statement.
- The break statement may or may not appear after each statement.

`switch` structure summary

- When a case statement is selected, code will execute until a **`break`** statement is found or the end of the switch structure is reached.

- If the value of the expression does not match any of the case values, the statements following the **`default`** label execute.
- If there is no **`default`** label, and if the value of the expression does not match any of the case values, the entire switch statement is skipped.

`switch` statement Rules

- The expression within the switch must evaluate to an integer.
- A constant expression must follow the case label (i.e. cannot use a variable).
- case labels must have different values.
- Different case statements may be followed by the same statements.
- The default label is not compulsory.
- There can be at most one default label.

So...

```
switch (x)
{
        case 1:     cout << "One" << endl;
        case 2:     cout << "Two" << endl;
        case 3:     cout << "Three" << endl;
```

What is the output of this switch statement if x=2?

Answer:

If **x** has the value 2, the output is:

Two

Three

Example

- Let's write ourselves a little integer desk calculator.

- The input will be
- a left operand
- an operator
- a right operand

- The output will be the result of the operation.