

Programming

What is programming?

- Series of instructions to a computer to accomplish a task
- Instructions must be written in a way the computer can understand
- Programming languages are used to write programs

What is programming?

- Once the code (language) of a program has been written, it must be executed (run, started).
- You may need to type the name of the program to start it, or use a word like RUN and the name of the program (in the old days, anyway).

How do you write a program?

- Decide what steps are needed to complete the task
- Write the steps in *pseudocode* (written in English) or as a *flowchart* (graphic symbols)
- Translate into the programming language
- Try out the program and “debug” it (fix if necessary)

What is pseudocode?

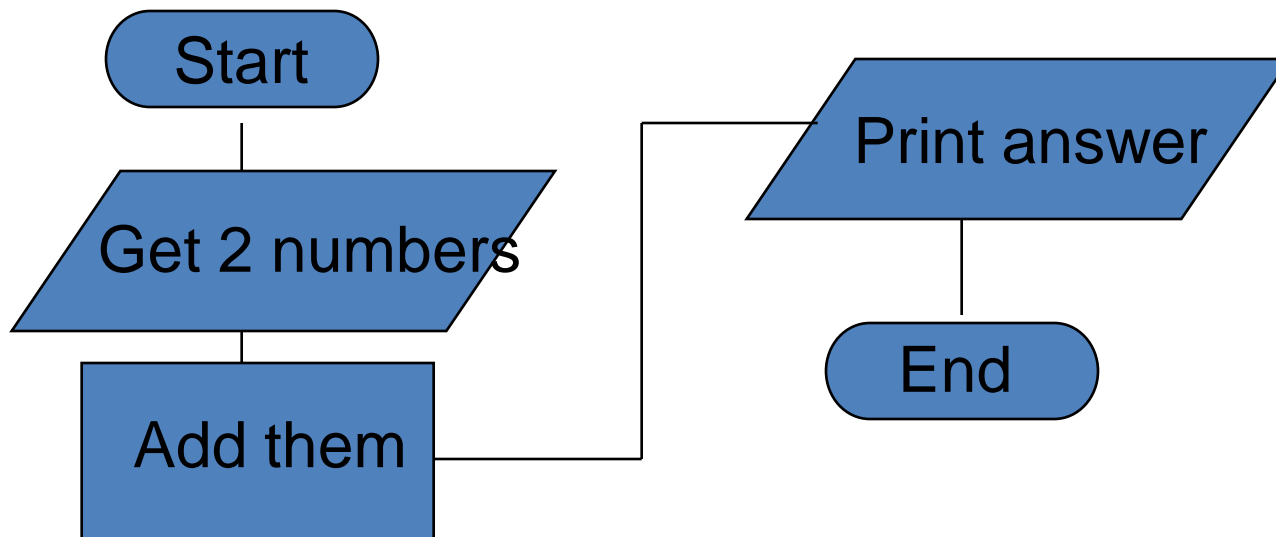
- List of steps written in English
- Like the instructions for a recipe
- Must be in the right sequence
 - Imagine saying “bake the cake” and then “mix it up”

Sample Pseudocode

- Task: add two numbers
- Pseudocode:
 - Start
 - Get two numbers
 - Add them
 - Print the answer
 - End

What does a flowchart look like?

- The pseudocode from the previous slide would look like this as a flowchart:



What are those funny symbols?

- START/END



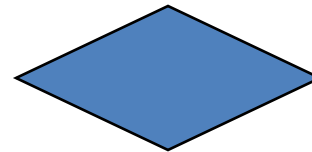
- INPUT/OUTPUT



- PROCESS



- DECISION



What are those funny symbols?

- START/END
- Used at the beginning and end of each flowchart.



What are those funny symbols?

- INPUT/OUTPUT
- Shows when information/data comes into a program or is printed out.



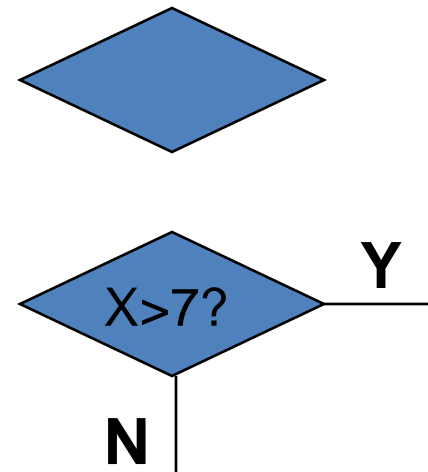
What are those funny symbols?

- PROCESS
- Used to show calculations, storing of data in variables, and other “processes” that take place within a program.



What are those funny symbols?

- DECISION
- Used to show that the program must decide whether something (usually a comparison between numbers) is true or false. YES and NO (or T/F) branches are usually shown.

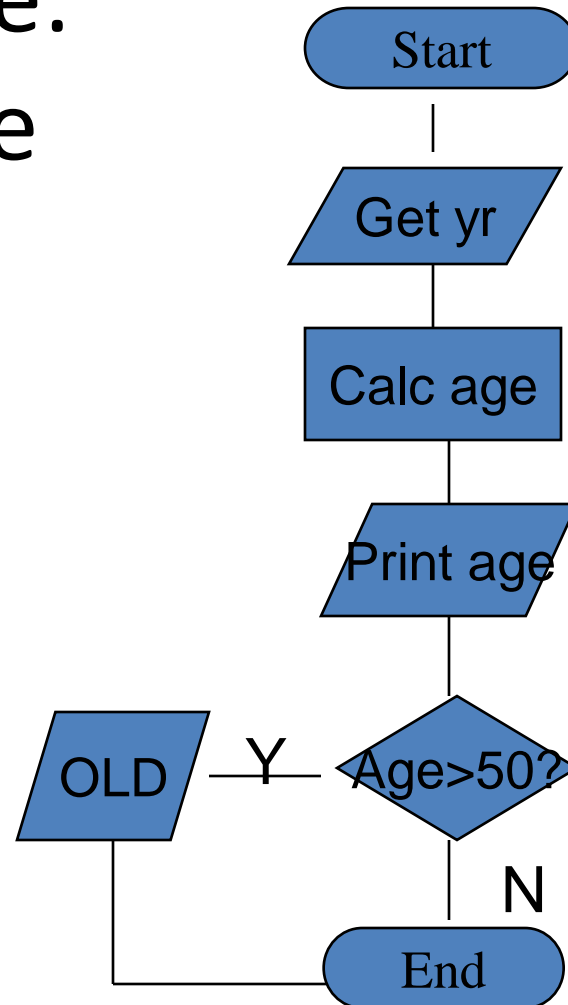


Another Sample: Calculating Age

- Pseudocode:
 - Start
 - Get year born
 - Calculate age
 - Print age
 - If age > 50 print OLD
 - End

Another Sample: Calculating Age

- Flowchart →
 - Start
 - Get year born
 - Calculate age
 - Print age
 - If age > 50 print OLD
 - End



Elements of a Program

- All programming languages have certain features in common. For example:
 - Variables
 - Commands/Syntax (the way commands are structured)
 - Loops
 - Decisions
 - Functions
- Each programming language has a different set of rules about these features.

Variables

- **Variables** are part of almost every program.
- A variable is a “place to put data” and is usually represented by a letter or a word. (Think of a variable as a Tupperware container with a label on it.)
- Variable names cannot contain spaces.
- Some programming languages have very specific limits on variable names.

Variables

- Usually there are several ways to put information into a variable.
- The most common way is to use the equal sign (=).
- $X = Y + 7$ means *take the value of Y, add 7, and put it into X.*
- $COUNT = COUNT + 2$ means *take the current value of COUNT, add 2 to it, and make it the new value of COUNT.*

Variables

- Sometimes you must specify the type of data that will be placed in a variable.
- Here are some examples of data types:
 - Numeric (numbers of all kinds)
 - String (text, “strings of letters”)
 - Integer (whole numbers)
 - Long (large numbers)
 - Boolean (true/false)

Variables

- Variables may be classified as *global* or *local*.
- A *global* variable is one that can be shared by all parts of a program, including any functions or sub-programs.
- A *local* variable is one that is used only within a certain part of the program, for example, only in one function or sub-program.

Commands/Syntax

- Programming languages are truly languages.
- They have rules about grammar, spelling, punctuation, etc.
- You need to learn the rules of a programming language, just as you learned to speak and write English.

Loops

- A **loop** is a repetition of all or part of the commands in a program.
- A loop often has a counter (a variable) and continues to repeat a specified number of times.
- A loop may also continue until a certain condition is met (e.g., until the end of a file or until a number reaches a set limit)

Decisions

- You saw a flowchart symbol for **decisions**.
- A program often needs to decide whether something is true or false in order to see which way to continue.
- Programs often use IF (or IF THEN or IF THEN ELSE) statements to show a decision.

Decisions

- An IF statement always has a condition to check, often a comparison between a variable and a number.
- The IF statement also must specify what to do if the condition/comparison is true.
- These instructions (for “true”) may come after the word THEN, or they may simply be listed.

Decisions

- In an IF THEN statement, when the condition is false, the program simply ignores the THEN commands and continues to the next line.
- In an IF THEN ELSE statement, commands are given for both the true and false conditions.

Functions

- In most programming languages, small sub-programs are used to perform some of the tasks.
- These may be called functions, subroutines, handlers, or other such terms.
- Functions often have names (e.g., getName or CALCTAX).

Functions

- A **function** generally gets information from the main program, performs some task, and returns information back to the program.
- Functions follow the same rules of syntax, etc. as the main program.

Hints for Writing Code

- “Code” means writing the program in the appropriate language
- Be sure the code is exact (spelling, capitals/lower case, punctuation, etc).
- Write part of the code, try it, then write more.

Debugging

- To “debug” means to try a program, then fix any mistakes.
- Virtually no program works the first time you run it. There are just too many places to make errors.
- When you are debugging a program, look for spelling and punctuation errors.
- Fix one error at a time, then try the program again.

3. C++ Stream Input/Output

- C++,
cout << "Enter new tag: ";
cin >> tag;
cout << "The new tag is : " << tag << '\n';

3.1 An Example

```
// Simple stream input/output
#include <iostream.h>

main()
{
    cout << "Enter your age: ";
    int myAge;
    cin >> myAge;

    cout << "Enter your friend's age: ";
    int friendsAge;
    cin >> friendsAge;
```

```
if (myAge > friendsAge)
    cout << "You are older.\n";
else
    if (myAge < friendsAge)
        cout << "You are younger.\n";
    else
        cout << "You and your friend are the same
age.\n";

return 0;
}
```

4. Declarations in C++

- In C++, declarations can be placed anywhere (except in the condition of a `while`, `do/while`, `for` or `if` structure.)
- An example

```
cout << "Enter two integers: ";  
int x, y;  
cin >> x >> y;  
cout << "The sum of " << x << " and " << y  
    << " is " << x + y << '\n';
```


Control Structures in C++

while, do/while, for
switch, break, continue

The **while** Repetition Structure

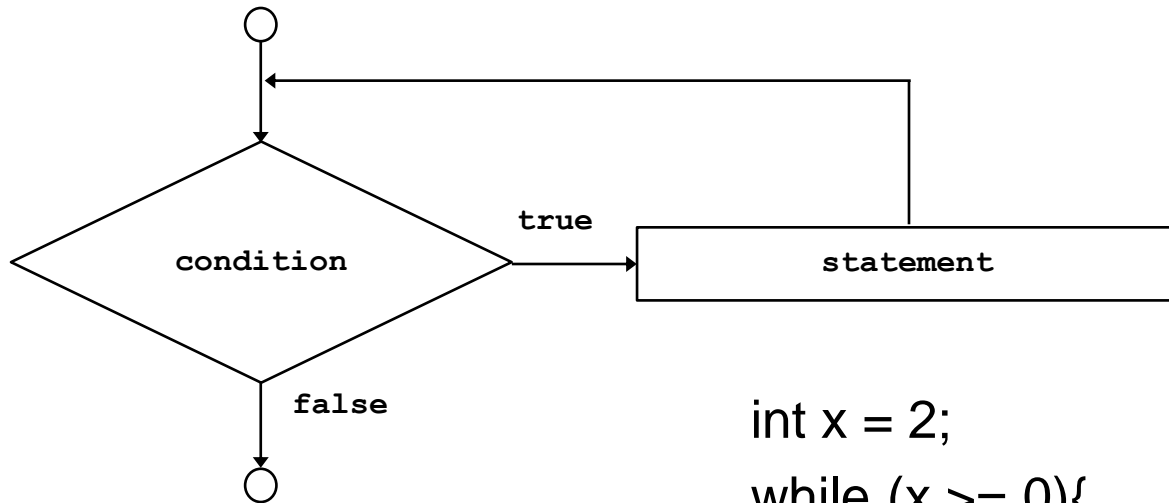
- Repetition structure
 - Programmer specifies an action to be repeated while some condition remains true
 - Psuedocode
 - while there are more items on my shopping list*
 - Purchase next item and cross it off my list*
 - **while** loop repeated until condition becomes false.

- Example

```
int product = 2;  
while ( product <= 1000 )  
    product = 2 * product;
```

The **while** Repetition Structure

- Flowchart of **while** loop



```
int x = 2;
while (x >= 0){
    if ( x == 2){
        cout << "Value of x is : " << x << endl;
    }
    x = x - 1;
}
```

- Common errors:
 - infinite loop
 - uninitialized variables

There are functions that return True or False :

`cin.eof()`

So..

```
char s;  
  
while (!cin.eof( )) {  
    cin >> s;  
    cout << s << endl;  
}
```

Formulating Algorithms (Counter-Controlled Repetition)

- Pseudocode for example:

Set total and grade counter to zero

While grade counter \leq 10

Input the next grade

Add the grade into the total

grade counter++

average = total divided / 10

Print the class average

- Following is the C++ code for this example

```

1 // Fig. 2.7: fig02_07.cpp
2 // Class average program with counter-controlled repetition
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int main()
10 {
11     int total,          // sum of grades
12         gradeCounter, // number of grades entered
13         grade,        // one grade
14         average;     // average of grades
15
16     // initialization phase
17     total = 0;                // clear total
18     gradeCounter = 1;        // prepare for loop
19
20     // processing phase
21     while ( gradeCounter <= 10 ) { // loop 10 times
22         cout << "Enter grade: "; // prompt for input
23         cin >> grade;           // input grade
24         total = total + grade;  // add grade to total
25         gradeCounter = gradeCounter + 1; // increment counter
26     }
27
28     // termination phase
29     average = total / 10;       // integer division
30     cout << "Class average is " << average << endl;
31
32     return 0; // indicate program ended successfully
33 }

```

The counter gets incremented each time the loop executes. Eventually, the counter causes the loop to end.

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

Program Output

Assignment Operators

- Assignment expression abbreviations

`c = c + 3;` can be abbreviated as `c += 3;` using the addition assignment operator

- Statements of the form

`variable = variable operator expression;`

can be rewritten as

`variable operator= expression;`

- Examples of other assignment operators include:

`d -= 4` (`d = d - 4`)

`e *= 5` (`e = e * 5`)

`f /= 3` (`f = f / 3`)

`g %= 9` (`g = g % 9`)

Increment and Decrement Operators

- Increment operator (`c++`) - can be used instead of

`c += 1`

- Decrement operator (`c--`) - can be used instead of

`c -= 1`

- Preincrement

- When the operator is used before the variable (`++c` or `--c`)
- Variable is changed, then the expression it is in is evaluated.

- Postincrement

- When the operator is used after the variable (`c++` or `c--`)
- Expression the variable is in executes, then the variable is changed.

- If `c = 5`, then
 - `cout << ++c;` prints out **6** (`c` is changed before `cout` is executed)
 - `cout << c++;` prints out **5** (`cout` is executed before the increment. `c` now has the value of **6**)

- When Variable is not in an expression
 - Preincrementing and postincrementing have the same effect.

```
++c;
```

```
cout << c;
```

and

```
c++;
```

```
cout << c;
```

have the same effect.

Essentials of Counter-Controlled Repetition

- Counter-controlled repetition requires:
 - The name of a control variable (or loop counter).
 - The initial value of the control variable.
 - The condition that tests for the final value of the control variable (i.e., whether looping should continue).
 - The increment (or decrement) by which the control variable is modified each time through the loop.
- Example:

```
int counter =1;           //initialization
while (counter <= 10){ //repetitio
// condition
    cout << counter << endl;
    ++counter;           //increment
}
```

The for Repetition Structure

- The general format when using **for** loops is

```
for ( initialization; LoopContinuationTest;
      increment )
    statement
```
- Example:

```
for( int counter = 1; counter <= 10;
    counter++ )
    cout << counter << endl;
```

 - Prints the integers from one to ten

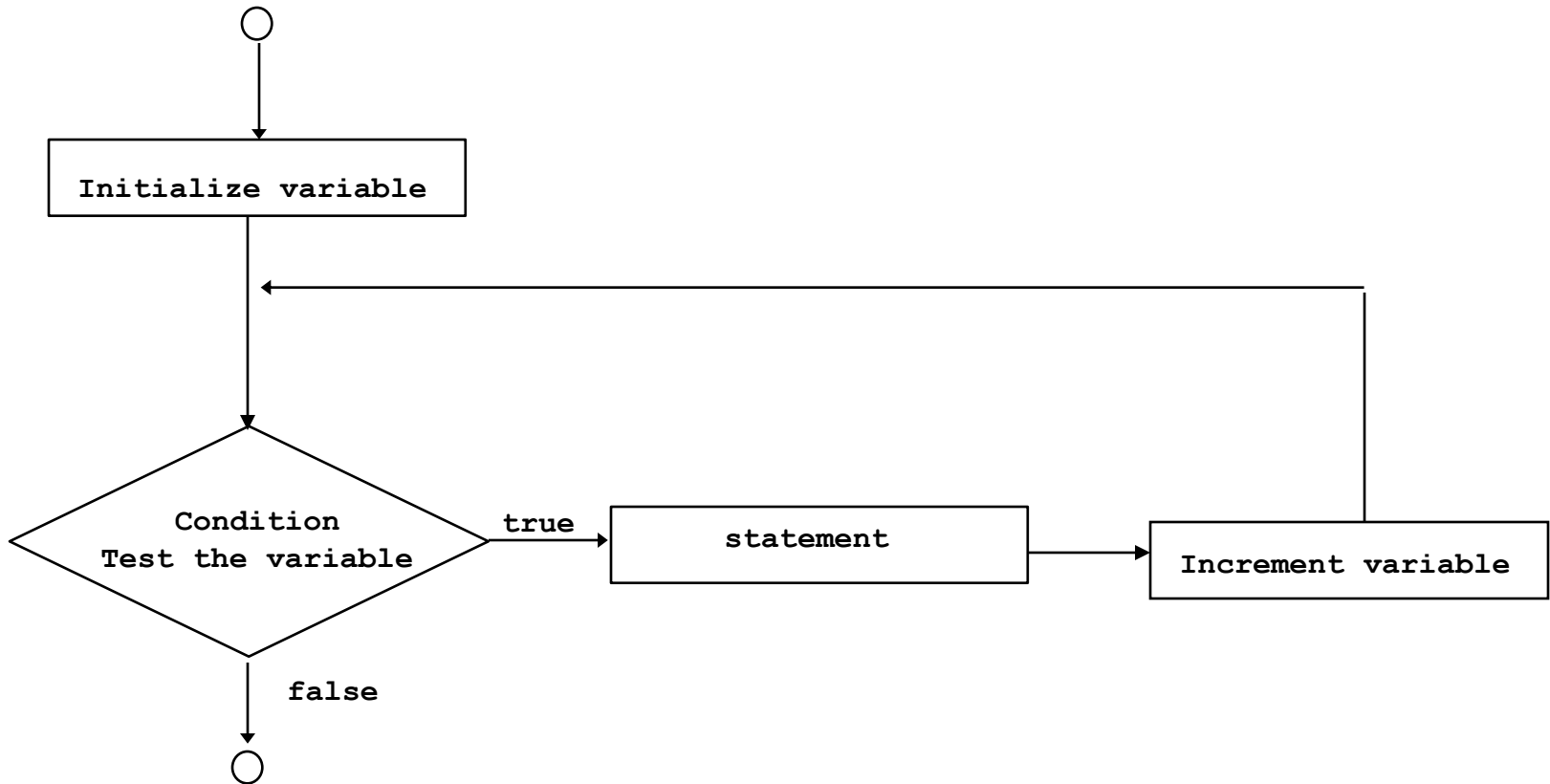
- **For** loops can usually be rewritten as **while** loops:

```
    initialization;
    while ( loopContinuationTest) {
        statement
        increment;
    }
```

- Initialization and increment as comma-separated lists

```
    for (int i = 0, j = 0;  j + i <= 10;
        j++, i++)
        cout << j + i << endl;
```

Flowchart for **for**



- Program to sum the even numbers from 2 to

```
100 1 // Fig. 2.20: fig02_20.cpp
    2 // Summation with for
    3 #include <iostream>
    4
    5 using std::cout;
    6 using std::endl;
    7
    8 int main()
    9 {
   10     int sum = 0;
   11
   12     for ( int number = 2; number <= 100; number += 2 )
   13         sum += number;
   14
   15     cout << "Sum is " << sum << endl;
   16
   17     return 0;
   18 }
```

Sum is 2550

The switch Multiple-Selection Structure

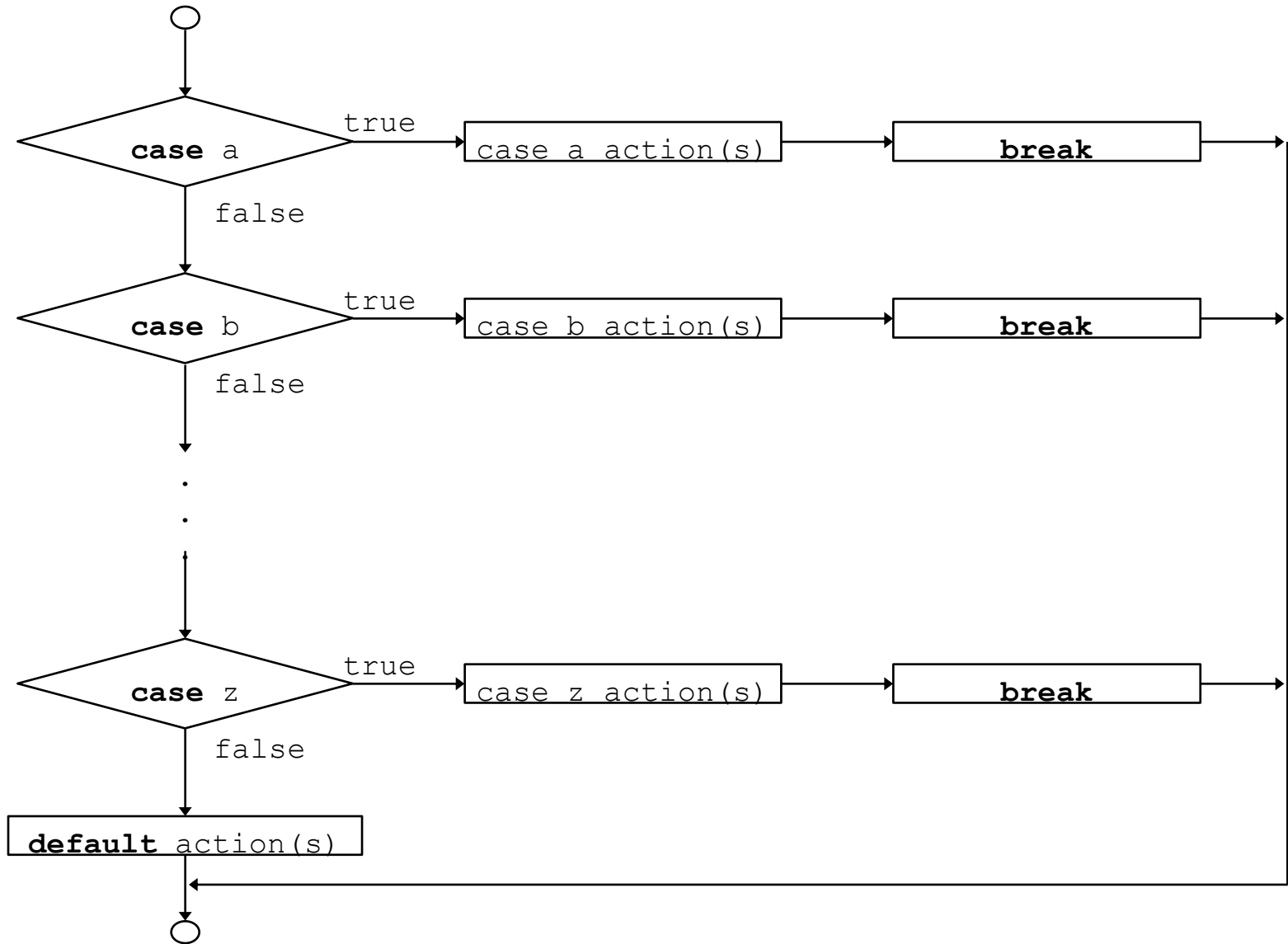
- **switch**
 - Useful when variable or expression is tested for multiple values
 - Consists of a series of **case** labels and an optional **default** case
 - **break** is (almost always) necessary

```
switch (expression) {  
    case val1:  
        statement  
        break;  
    case val2:  
        statement  
        break;  
    ....  
    case valn:  
        statement  
        break;  
    default:  
        statement  
        break;  
}
```



```
if (expression == val1)  
    statement  
else if (expression==val2)  
    statement  
....  
else if (expression== valn)  
    statement  
else  
    statement
```

flowchart



```

1 // Fig. 2.22: fig02_22.cpp
2 // Counting letter grades
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int main()
10 {
11     int grade,          // one grade
12         aCount = 0,    // number of A's
13         bCount = 0,    // number of B's
14         cCount = 0,    // number of C's
15         dCount = 0,    // number of D's
16         fCount = 0;    // number of F's
17
18     cout << "Enter the letter grades." << endl
19          << "Enter the EOF character to end input." << endl;
20
21     while ( ( grade = cin.get() ) != EOF ) {
22
23         switch ( grade ) {
24
25             case 'A': // grade was uppercase A
26             case 'a': // or lowercase a
27                 ++aCount;
28                 break; // necessary to exit switch
29
30             case 'B': // grade was uppercase B
31             case 'b': // or lowercase b
32                 ++bCount;
33                 break;
34

```

Notice how the case statement is used

```

35     case 'C': // grade was uppercase C
36     case 'c': // or lowercase c
37         ++cCount;
38         break;
39
40     case 'D': // grade was uppercase D
41     case 'd': // or lowercase d
42         ++dCount;
43         break;
44
45     case 'F': // grade was uppercase F
46     case 'f': // or lowercase f
47         ++fCount;
48         break;
49
50     case '\n': // ignore newlines,
51     case '\t': // tabs,
52     case ' ': // and spaces in
53         break;
54
55     default: // catch all other characters
56         cout << "Incorrect letter grade entered."
57             << " Enter a new grade." << endl;
58         break; // optional
59 }
60 }
61
62 cout << "\n\nTotals for each letter grade are:"
63     << "\nA: " << aCount
64     << "\nB: " << bCount
65     << "\nC: " << cCount
66     << "\nD: " << dCount
67     << "\nF: " << fCount << endl;
68
69 return 0;
70 }

```

break causes switch to end and the program continues with the first statement after the switch structure.

Notice the default statement.

Enter the letter grades.

Enter the EOF character to end input.

a

B

c

C

A

d

f

C

E

Incorrect letter grade entered. Enter a new grade.

D

A

b

Totals for each letter grade are:

A: 3

B: 2

C: 3

D: 2

F: 1

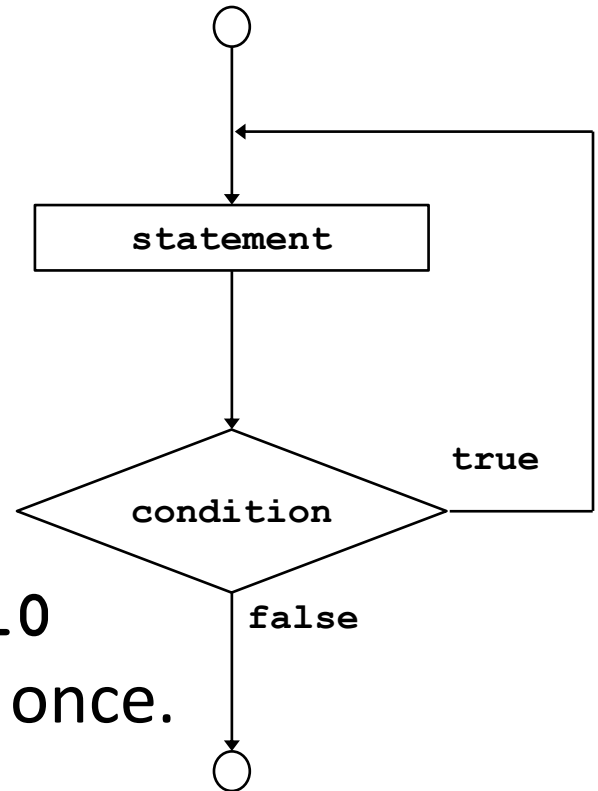
The do/while Repetition Structure

- The **do/while** repetition structure is similar to the **while** structure,
 - Condition for repetition tested after the body of the loop is executed
- Format:

```
do {  
    statement  
} while ( condition );
```
- Example (letting counter = 1):

```
do {  
    cout << counter << " ";  
} while (++counter <= 10);
```

 - This prints the integers from 1 to 10
- All actions are performed at least once.



The **break** and **continue** Statements

- **Break**

- Causes immediate **exit** from a **while**, **for**, **do/while** or **switch** structure
- Program execution continues with the first statement after the structure
- Common uses of the **break** statement:
 - Escape early from a loop
 - Skip the remainder of a **switch** structure

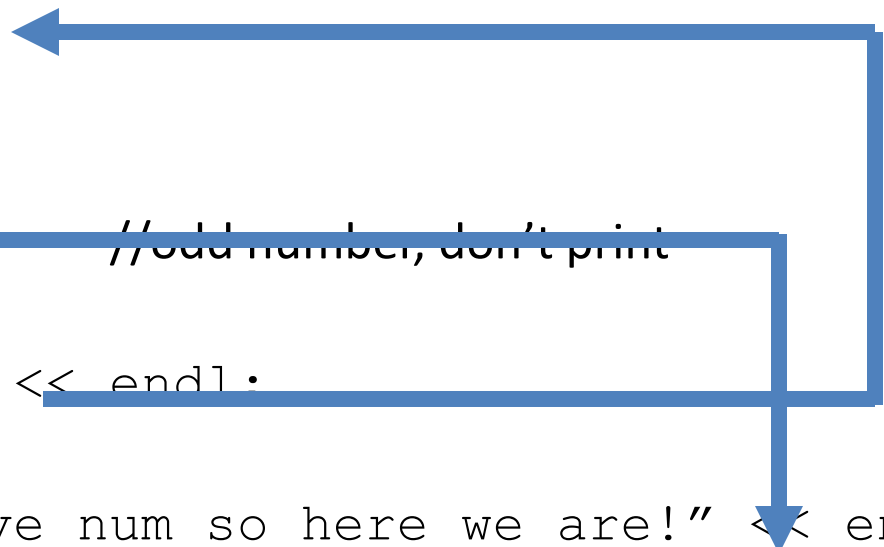
- **Continue**

- Skips the remaining statements in the body of a **while**, **for** or **do/while** structure and proceeds with the next iteration of the loop
- In **while** and **do/while**, the loop-continuation test is evaluated immediately after the **continue** statement is executed
- In the **for** structure, the increment expression is executed, then the loop-continuation test is evaluated

The continue Statement

- Causes an immediate jump to the loop test

```
int next = 0;
while (true){
    cin >> next;
    if (next < 0)
        break;
    if (next % 2) //odd number, don't print
        continue;
    cout << next << endl;
}
cout << "negative num so here we are!" << endl;
```



Sentinel-Controlled Repetition

- Suppose the previous problem becomes:
 - Develop a class-averaging program that will process an arbitrary number of grades each time the program is run.*
 - Unknown number of students - how will the program know to end?
- Sentinel value
 - Indicates “end of data entry”
 - Loop ends when sentinel inputted
 - Sentinel value chosen so it cannot be confused with a regular input (such as -1 in this case)

- Top-down, stepwise refinement
 - begin with a pseudocode representation of the top:

Determine the class average for the quiz

- Divide top into smaller tasks and list them in order:

Initialize variables

Input, sum and count the quiz grades

Calculate and print the class average

Input, sum and count the quiz grades

to

Input the first grade (possibly the sentinel)

While the user has not as yet entered the sentinel

Add this grade into the running total

Add one to the grade counter

Input the next grade (possibly the sentinel)

- **Refine**

Calculate and print the class average

to

If the counter is not equal to zero

Set the average to the total divided by the counter

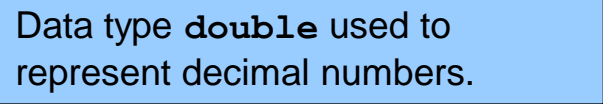
Print the average

Else

Print "No grades were entered"

```
1 // Fig. 2.9: fig02_09.cpp
2 // Class average program with sentinel-controlled repetition.
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8 using std::ios;
9
10 #include <iomanip>
11
12 using std::setprecision;
13 using std::setiosflags;
14
15 int main()
16 {
17     int total,           // sum of grades
18         gradeCounter, // number of grades entered
19         grade;          // one grade
20     double average;    // number with decimal point for average
21
22     // initialization phase
23     total = 0;
24     gradeCounter = 0;
25
26     // processing phase
27     cout << "Enter grade, -1 to end: ";
28     cin >> grade;
29
30     while ( grade != -1 ) {
```

Data type `double` used to represent decimal numbers.



Nested control structures

- Problem:

A college has a list of test results (1 = pass, 2 = fail) for 10 students. Write a program that analyzes the results. If more than 8 students pass, print "Raise Tuition".

- We can see that

- The program must process 10 test results. A counter-controlled loop will be used.
- Two counters can be used—one to count the number of students who passed the exam and one to count the number of students who failed the exam.
- Each test result is a number—either a 1 or a 2. If the number is not a 1, we assume that it is a 2.

Nested control structures

- High level description of the algorithm

Initialize variables

Input the ten quiz grades and count passes and failures

*Print a summary of the exam results and decide if
tuition should be raised*


```
1 // Fig. 2.11: fig02_11.cpp
2 // Analysis of examination results
3 #include <iostream>
4
5 using std::cout;
6 using std::cin;
7 using std::endl;
8
9 int main()
10 {
11     // initialize variables in declarations
12     int passes = 0,           // number of passes
13         failures = 0,       // number of failures
14         studentCounter = 1, // student counter
15         result;             // one exam result
16
17     // process 10 students; counter-controlled loop
18     while ( studentCounter <= 10 ) {
19         cout << "Enter result (1=pass,2=fail): ";
20         cin >> result;
21
22         if ( result == 1 )           // if/else nested in while
23             passes = passes + 1;
```

```

24     else
25         failures = failures + 1;
26
27     studentCounter = studentCounter + 1;
28 }
29
30 // termination phase
31 cout << "Passed " << passes << endl;
32 cout << "Failed " << failures << endl;
33
34 if ( passes > 8 )
35     cout << "Raise tuition " << endl;
36
37 return 0;    // successful termination
38 }

```

```

Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 2
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Enter result (1=pass,2=fail): 1
Passed 9
Failed 1
Raise tuition

```