ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

Lecture 03

CONTROL FLOW

Writing a program

- Before writing a program:
  - Have a thorough understanding of problem
  - Carefully planned approach for solving it
- While writing a program:
  - Know what “building blocks” are available
  - Use good programming principles

Algorithms

- Computing problems
  - All can be solved by executing a series of actions in a specific order
- Algorithm: procedure in terms of
  - *Actions* to be executed
  - *Order* in which these actions are to be executed
- Program control
  - Specify order in which statements are to executed

Pseudocode

- Pseudocode
  - Artificial, informal language that helps us develop algorithms
  - Similar to everyday English
  - Not actually executed on computers
  - Helps us “think out” a program before writing it
    - Easy to convert into a corresponding C program
    - Consists only of executable statements
  - Example: Recipes!

Control Structures

- Sequence execution
  - Statements executed one after the other in the order written
- Selection structures:
  - if, if/else, and switch
- Repetition (Loops) structures:
  - while, for and do/while
- Transfer controls:
  - goto, breaks, continues

Control Structures (II)

- Flowchart
  - Graphical representation of an algorithm
  - Drawn using certain special-purpose symbols connected by arrows called *flowlines.*
  - Rectangle symbol (action symbol): indicates any type of action.
  - Oval symbol: indicates beginning or end of a program, or a section of code (circles).
- Single-entry/single-exit control structures
  - Connect exit point of one control structure to entry point of the next (*control-structure stacking*).
  - Makes programs easy to build

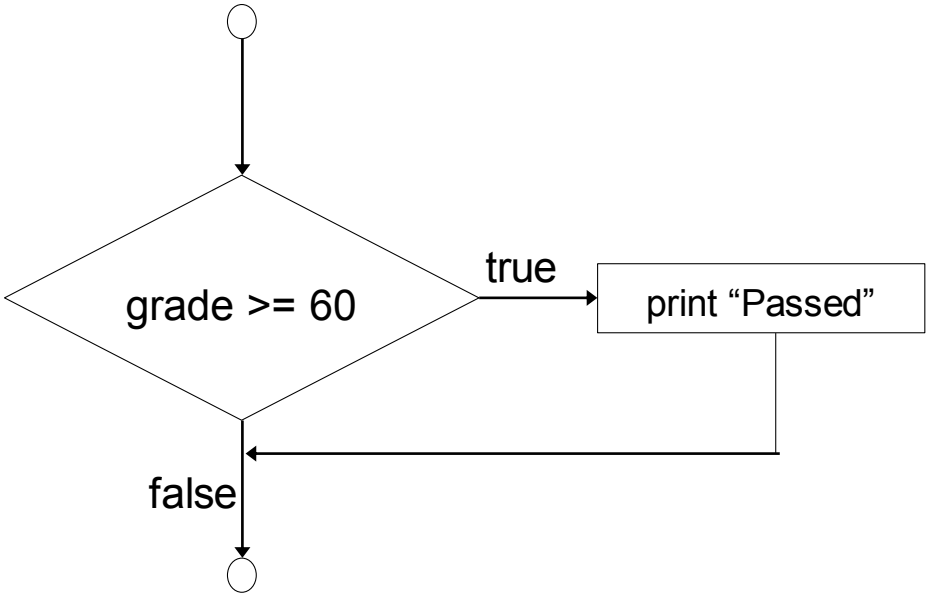
The `if` Selection Structure

- Selection structure:
  - Used to choose among alternative courses of action
  - Pseudocode: *If student's grade is greater than or equal to 60*
    Print "Passed"
- If condition **`true`**
  - Print statement executed and program goes on to next statement.
  - If **`false`**, print statement is ignored and the program goes onto the next statement.
  - Indenting makes programs easier to read
    - C ignores whitespace characters.
- Pseudocode statement in C:

```
if ( grade >= 60 )
   printf( "Passed\n" );
```

The `if` Selection Structure (II)

- Diamond symbol (decision symbol) - indicates decision is to be made
  - Contains an expression that can be **`true`** or **`false`**
  - Test the condition, follow appropriate path
- **`if`** structure is a single-entry/single-exit structure.



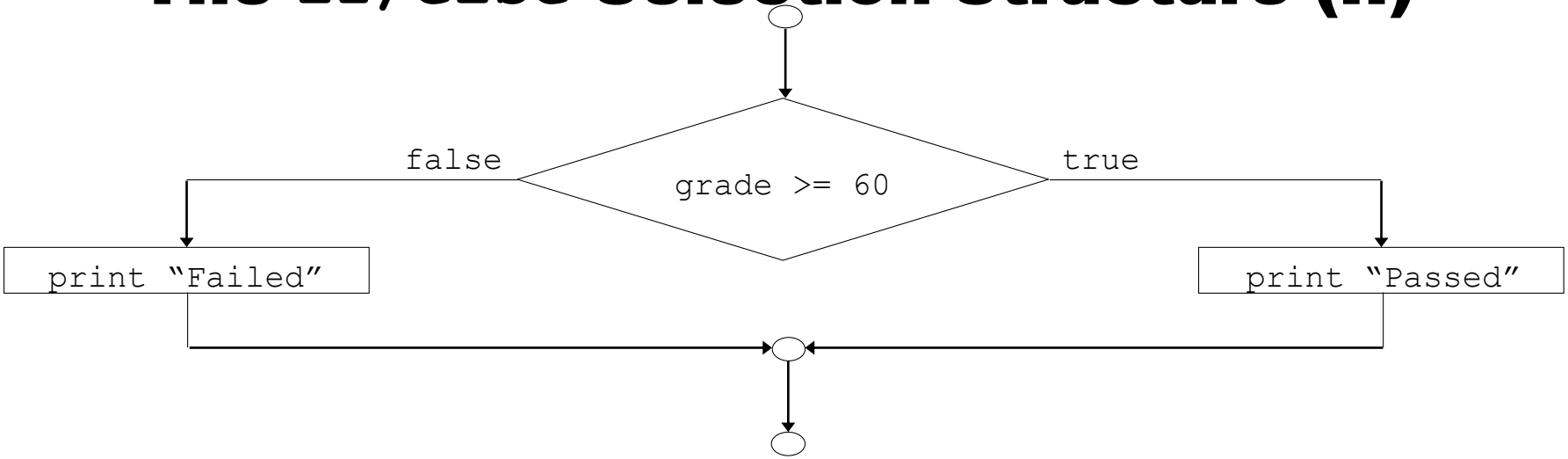
A decision can be made on any expression.
zero - **`false`**
nonzero - **`true`**
Example:
`3 - 4` is **`true`**

The `if/else` Selection Structure

- **`if`**
  - Only performs an action if the condition is **`true`**.
- **`if/else`**
  - A different action when condition is **`true`** than when condition is **`false`**
- Psuedocode: *If student's grade is greater than or equal to 60*
  Print "Passed"
  else
  Print "Failed"
  - Note spacing/indentation conventions
- C code:

```
if ( grade >= 60 )
   printf( "Passed\n");
else
   printf( "Failed\n");
```

The `if/else` Selection Structure (II)



- Ternary conditional operator (`?:`)
  - Takes three arguments (condition, value if **`true`**, value if **`false`**)
  - Our pseudocode could be written:

```
printf( "%s\n", grade >= 60 ? "Passed" : "Failed" );
```

OR

```
grade >= 60 ? printf( "Passed\n" ) : printf( "Failed\n" );
```

The `if/else` Selection Structure (III)

- Nested **`if/else`** structures
  - Test for multiple cases by placing **`if/else`** selection structures inside **`if/else`** selection structures

```
If student's grade is greater than or equal to 90
   Print "A"
else
   If student's grade is greater than or equal to 80
                Print "B"
   else
                If student's grade is greater than or equal to 70
                             Print "C"
                else
                             If student's grade is greater than or equal to 60
                                          Print "D"
                             else
                                          Print "F"
```

  - Once condition is met, rest of statements skipped
  - Deep indentation usually not used in practice

The `if/else` Selection Structure (IV)

- Compound statement:
  - Set of statements within a pair of braces
  - Example:

```
if ( grade >= 60 )
   printf( "Passed.\n" );
else {
   printf( "Failed.\n" );
   printf( "You must take this course again.\n" );
}
```

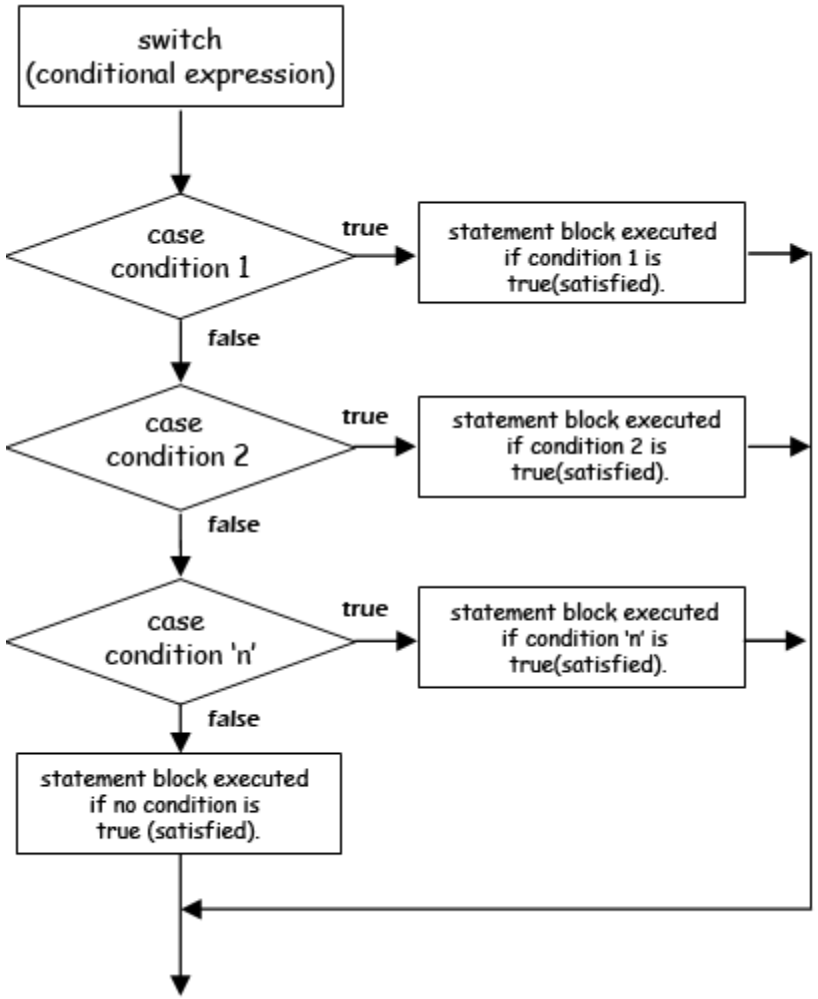
  - Without the braces,

```
printf( "You must take this course again.\n" );
```

    would be automatically executed
- Block: compound statements with declarations

The `Switch` Selection Structure

- The `switch` statement is a multi-way decision that tests whether an expression matches one of a number of **constant integer** values



The `Switch` Selection Structure (II)

- switch (*expression*) {
  case *const-expr*: *statements*
  case *const-expr*: *statements*
  default: *statements*
  }
- The `break` statement causes an immediate exit from the switch.

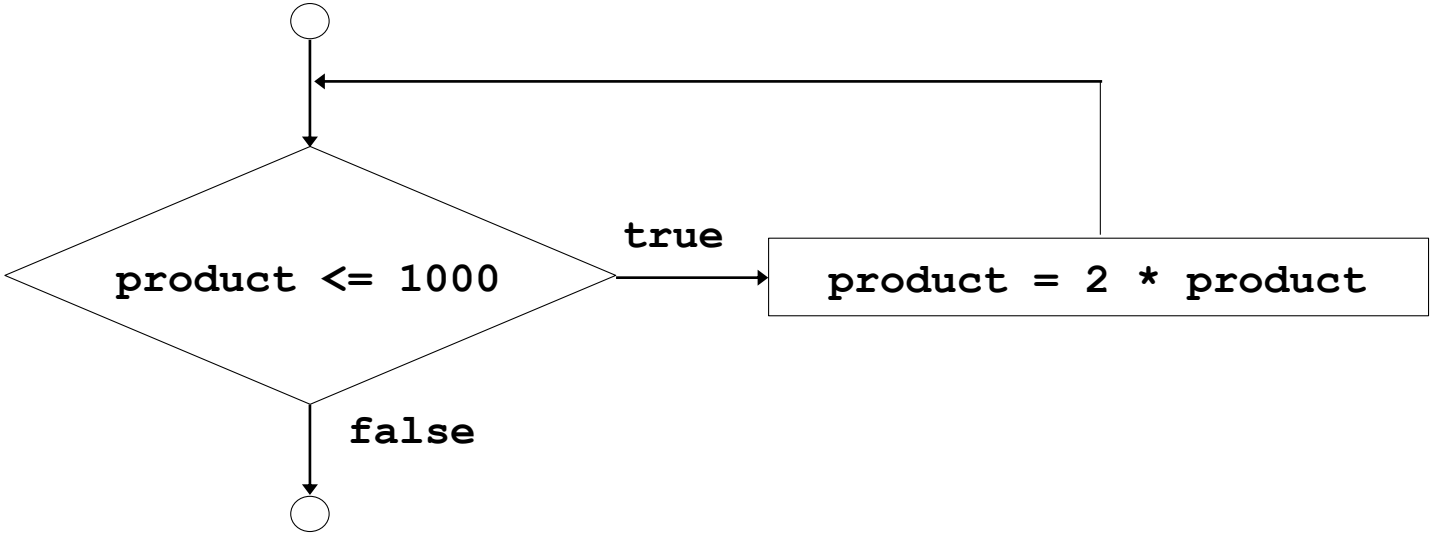
The `while` Repetition Structure

- Repetition structure
  - Programmer to specify 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`**

The `while` Repetition Structure (II)

- Example:

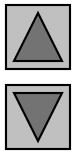
```
int product = 2;
while ( product <= 1000 )
   product = 2 * product;
```



Formulating Algorithms (Counter-Controlled Repetition)

- Counter-controlled repetition
  - Loop repeated until counter reaches a certain value.
  - Definite repetition: number of repetitions is known
  - Example: *A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.*
- Pseudocode:

```
Set total to zero
    Set grade counter to one
While grade counter is less than or equal to ten
        Input the next grade
        Add the grade into the total
        Add one to the grade counter
Set the class average to the total divided by ten
    Print the class average
```



Outline

1. **Initialize Variables**
2. **Execute Loop**
3. **Output results**

```c
/* Fig. 3.6: fig03_06.c
   Class average program with
   counter-controlled repetition */
#include <stdio.h>

int main()
{
   int counter, grade, total, average;

   /* initialization phase */
   total = 0;
   counter = 1;

   /* processing phase */
   while ( counter <= 10 ) {
      printf( "Enter grade: " );
      scanf( "%d", &grade );
      total = total + grade;
      counter = counter + 1;
   }

   /* termination phase */
   average = total / 10;
   printf( "Class average is %d\n", average );

   return 0;   /* indicate program ended successfully */
}
```

Sentinel-Controlled Repetition

- 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?

- Use sentinel value

  – Also called signal value, dummy value, or flag 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

- 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
- Many programs have three phases
  - Initialization: initializes the program variables
  - Processing: inputs data values and adjusts program variables accordingly
  - Termination: calculates and prints the final results
  - This Helps the breakup of programs for top-down refinement

Top-Down, Stepwise Refinement (II)

- Refine the initialization phase from *Initialize variables* to:

  Initialize total to zero

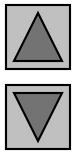
  Initialize counter to zero

- Refine *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"



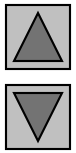
Outline

1. Initialize Variables

2. Get user input

2.1 Perform Loop

```
1  /* Fig. 3.8: fig03_08.c
2     Class average program with
3     sentinel-controlled repetition */
4  #include <stdio.h>
5
6  int main()
7  {
8     float average;              /* new data type */
9     int counter, grade, total;
10
11    /* initialization phase */
12    total = 0;
13    counter = 0;
14
15    /* processing phase */
16    printf( "Enter grade, -1 to end: " );
17    scanf( "%d", &grade );
18
19    while ( grade != -1 ) {
20       total = total + grade;
21       counter = counter + 1;
22       printf( "Enter grade, -1 to end: " );
23       scanf( "%d", &grade );
24    }
```



Outline

3. Calculate Average

3.1 Print Results

```
25
26    /* termination phase */
27    if ( counter != 0 ) {
28       average = ( float ) total / counter;
29       printf( "Class average is %.2f", average );
30    }
31    else
32       printf( "No grades were entered\n" );
33
34    return 0;    /* indicate program ended successfully */
35 }
```

Program Output

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

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"
- Notice that
  - The program must process 10 test results
    - Counter-controlled loop will be used
  - Two counters can be used
    - One for number of passes, one for number of fails
  - 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

- Top level outline

  Analyze exam results and decide if tuition should be raised

- First Refinement

  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

- Refine *Initialize variables* to

  Initialize passes to zero
  Initialize failures to zero
  Initialize student counter to one

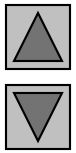
Nested control structures (III)

- Refine *Input the ten quiz grades and count passes and failures* to

```
While student counter is less than or equal to ten
    Input the next exam result
    If the student passed
        Add one to passes
    else
        Add one to failures
    Add one to student counter
```

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

```
Print the number of passes
Print the number of failures
If more than eight students passed
    Print "Raise tuition"
```



Outline

1. Initialize variables

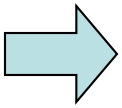
2. Input data and count passes/failures

3. Print results

```c
/*
   Analysis of examination results */
#include <stdio.h>

int main()
{
   /* initializing variables in declarations */
   int passes = 0, failures = 0, student = 1, result;

   /* process 10 students; counter-controlled loop */
   while ( student <= 10 ) {
      printf( "Enter result ( 1=pass,2=fail ): " );
      scanf( "%d", &result );

      if ( result == 1 )        /* if/else nested in while */
         passes = passes + 1;
      else
         failures = failures + 1;

      student = student + 1;
   }

   printf( "Passed %d\n", passes );
   printf( "Failed %d\n", failures );

   if ( passes > 8 )
      printf( "Raise tuition\n" );

   return 0;   /* successful termination */
}
```

The “for” loop (shorthand of such while loop)

```
float pow(float x, uint
exp)
{
  float result=1.0;
  int i;
  i=0;
  while (i < exp) {
    result = result * x;
    i++;
  }
  return result;
}

int main(int argc, char
**argv)
{
  float p;
  p = pow(10.0, 5);
  printf("p = %f\n", p);
  return 0;
}
```



```
float pow(float x, uint
exp)
{
  float result=1.0;
  int i;
  for (i=0; i < exp); i++)
{
    result = result * x;
  }
  return result;
}

int main(int argc, char
**argv)
{
  float p;
  p = pow(10.0, 5);
  printf("p = %f\n", p);
  return 0;
}
```

The `for` loop structure (II)

- In a for statement, it is possible to place multiple expressions in the various parts
- `continue` is used to causes the next iteration of the enclosing `for`

```
for (i = 0, j = strlen(s)-1;
i < j; i++, j--)
    c = s[i], s[i] = s[j],
s[j] = c;
```

```
for (i = 0; i < n; i++)
    if (a[i] < 0) /* skip
negative elements */
        continue;
    ... /* do positive
elements */
```