4

C How to Program

Selection Structure

Outline

4.1 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

The `if` Selection Structure

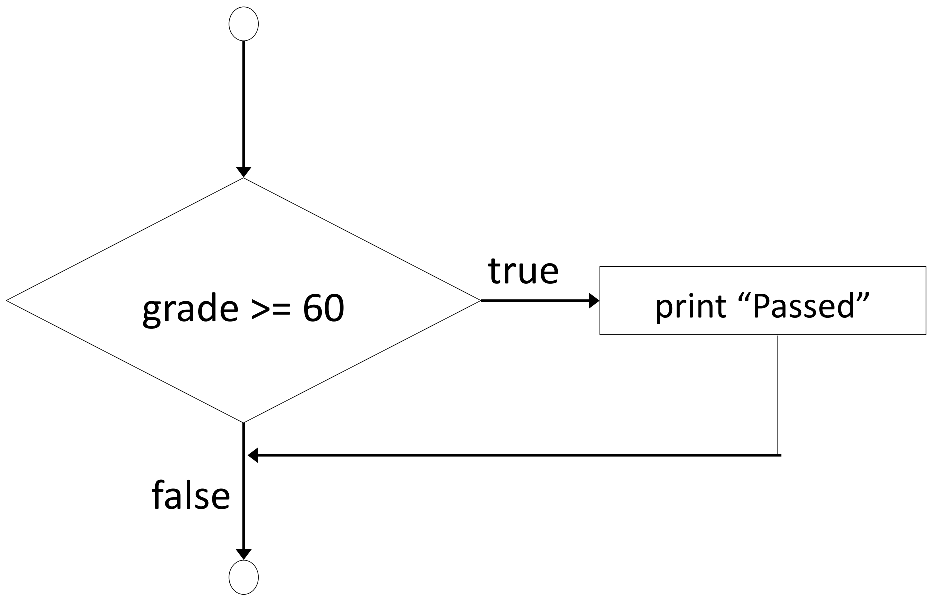
- Pseudocode statement in C:

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

  – C code corresponds closely to the pseudocode
- 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

The `if` Selection Structure

- `if` structure is a single-entry/single-exit structure



4.2 The `if/else` Selection Structure

- **`if`**
  - Only performs an action if the condition is **`true`**
- **`if/else`**
  - Specifies an action to be performed both when the condition is **`true`** and when it is **`false`**
- Psuedocode:

```
If student's grade is greater than or equal to 60
    Print "Passed"
else
    Print "Failed"
```

  - Note spacing/indentation conventions

The `if`/`else` Selection Structure

- C code:

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

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

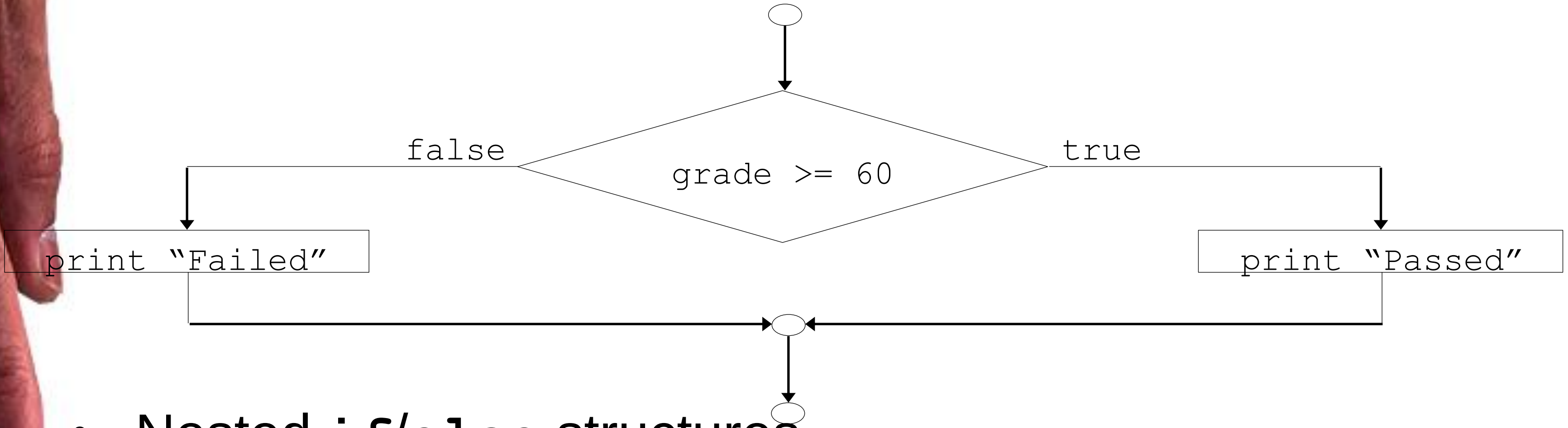
```
printf( "%s\n", grade >= 60 ? "Passed"
  : "Failed" );
```

  - Or it could have been written:

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

The **`if/else`** Selection Structure

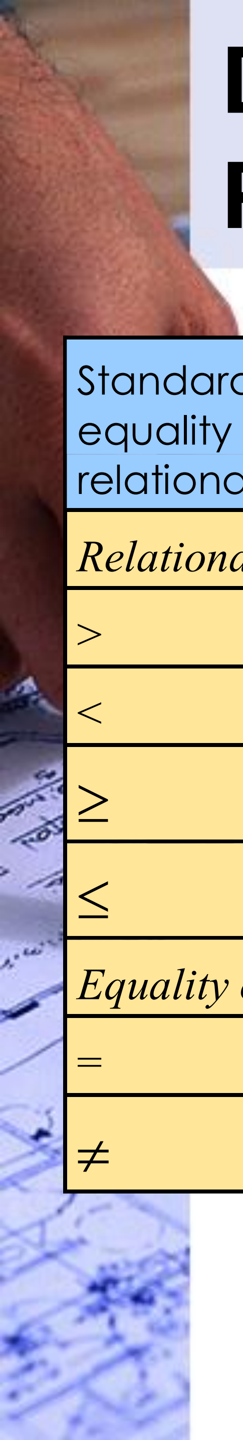
- Flow chart of the **`if/else`** selection structure



- Nested **`if/else`** structures
  - Test for multiple cases by placing **`if/else`** selection structures inside **`if/else`** selection structures
  - Once condition is met, rest of statements **skipped**
  - Deep indentation usually not used in practice

Decision Making: Equality and Relational Operators

- Equality and relational operators
  - Lower precedence than arithmetic operators
  - Equality operators
    - Same level of precedence
  - Relational operators
    - Same level of precedence
  - Associate left to right



Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Relational operators			
$>$	`>`	`x > y`	**x** is greater than **y**
$<$	`<`	`x < y`	**x** is less than **y**
$\geq$	`>=`	`x >= y`	**x** is greater than or equal to **y**
$\leq$	`<=`	`x <= y`	**x** is less than or equal to **y**
Equality operators			
$=$	`==`	`x == y`	**x** is equal to **y**
$\neq$	`!=`	`x != y`	**x** is not equal to **y**

Notice:

“==“

(not “=“)

The `if`/`else` Selection Structure

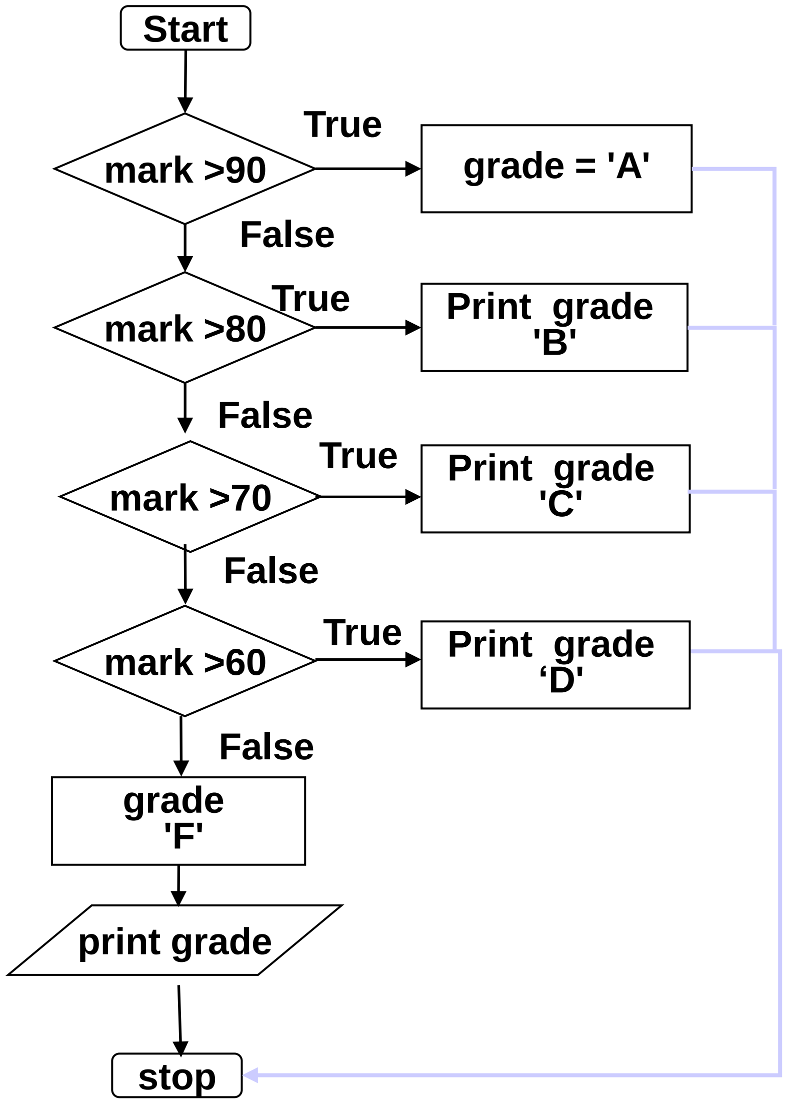
– Pseudocode for a nested `if`/`else` structure

```
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"
```

The `if/else` Selection Structure

```
int mark;
Printf("Enter your mark");
Scanf("%d", &mark);
  char grade;
    if (mark>90)
       grade='A';
    else if (mark>80)
       grade='B';
    else if (mark>70)
       grade='C';
    else if (mark>60)
       grade='D';
    else
       grade='F';

Printf("your grade is %d", grade);
```



The if/else Selection Structure

- 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, the statement

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

  would be executed automatically

4.3 The `switch` Multiple-Selection Structure

- **`switch`**
  - Useful when a variable or expression is tested for all the values it can be assumed and different actions are taken
- Format
  - Series of **`case`** labels and an optional **`default`** case

```
switch ( value ){
    case '1':
        actions
    case '2':
        actions
    default:
        actions
    }
```

  - **`break;`** exits from structure

How does switch **Structure work**

- It works in the following way:

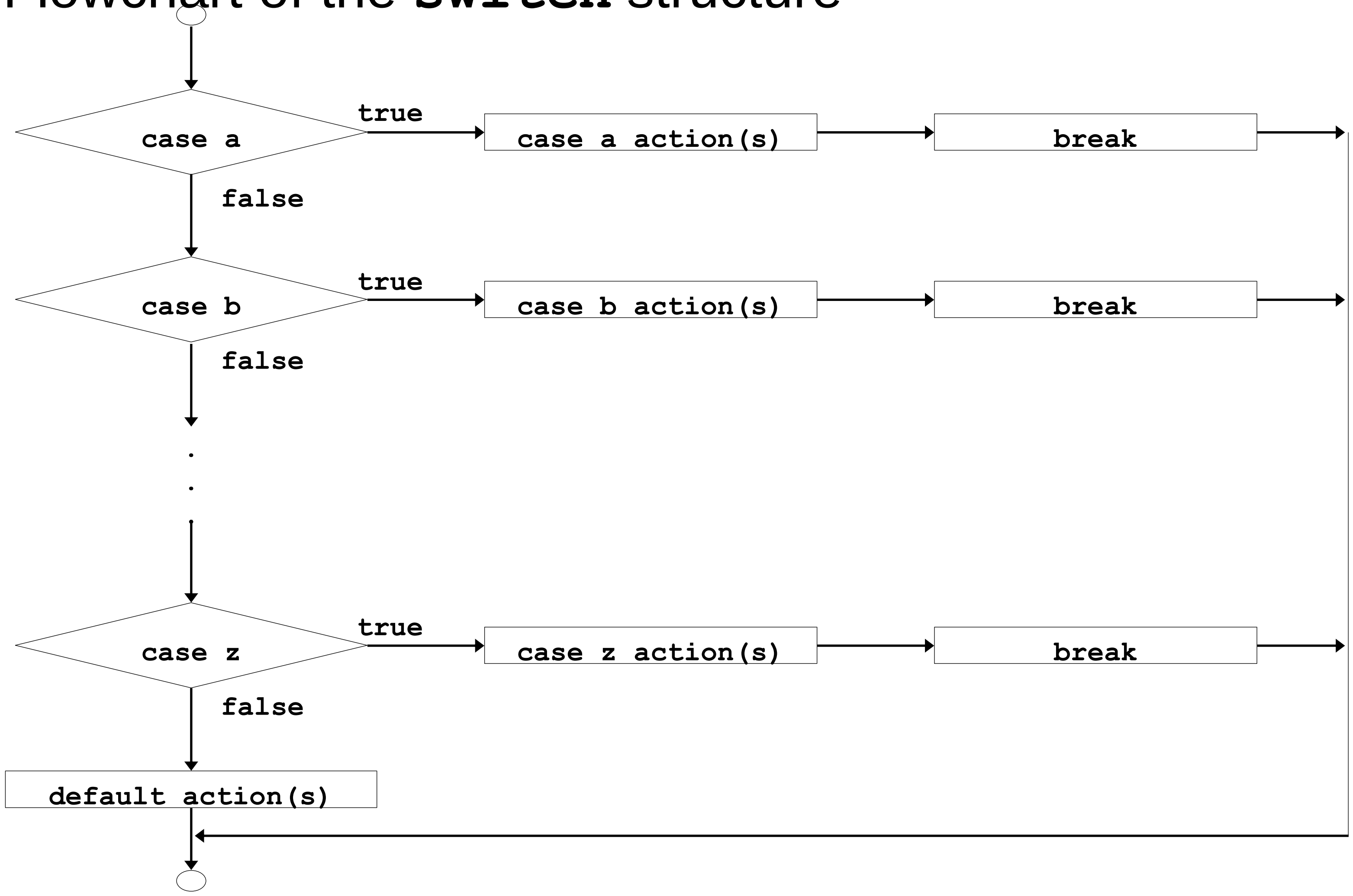
1. **switch** evaluates *expression* and checks if it is equivalent to *constant1*,
2. if it is, it executes *block of instructions 1* until it finds the **break** keyword, moment at which the program will jump to the end of the *switch* selective structure.
3. If *expression* was not equal to *constant1* it will check if *expression* is equivalent to *constant2*. If it is, it will execute *block of instructions 2* until it finds the **break** keyword.

How does switch Structure work

4. if the value of expression has not matched any of the previously specified constants (you may specify as many case sentences as values you want to check), the program will execute the instructions included in the **default**: section, if this one exists, since it is **optional**.
5. Notice that **switch** can only be used to compare an expression with different constants. Thus we cannot put variables (**case (n*2):**) or ranges (**case (1..3):**) because they are not valid constants.
6. If you need to check ranges or values that are not constants use a concatenation of **if** and **else if** sentences.

The `switch` Multiple-Selection Structure

- Flowchart of the **`switch`** structure



Switch Statement

```
switch (expression) {
    case const_expr1;
        statement1;
        break;
    case const_expr2;
        statement2;
        break;
    …
    default:
        statementn;
}
```

Example

```
switch (i) {
    case 1: printf("*");
        break;
    case 2: printf("**");
        break;
    case 3: printf("***");
        break;
    case 4: printf("****");
        break;
    case 5: printf("*****");
        break;
}
```

switch example

```
int x;
printf(“Enter a number”);
scanf(“%d”,&x);
switch (x) {
        case 1:
                    printf(“x is 1\n“);
                    break;
         case 2:
                    printf( "x is 2\n“);
                     break;
        default: printf( "value of x unknown\n“); }
```

if-else equivalent

```
int x;
printf(“Enter a number”);
scanf(“%d”,&x);
•  if (x == 1)
            printf(“x is 1\n“);
   else if (x == 2)
                  printf(“ "x is 2\n“);
            else
                  printf(“value of x unknown\n“);
```

switch example

```
int x;
printf(“Enter a number\n”);
scanf(“%d”,&x);
switch (x) {
case 1:
case 2:
case 3:
        printf("x is 1, 2 or 3\n“);
         break;
    default: printf ("x is not 1, 2 nor 3\n“); }
```

4.4 Logical Operators

- **`&&`** (logical AND)
  - Returns **`true`** if both conditions are **`true`**
- **`||`** (logical OR)
  - Returns **`true`** if either of its conditions are **`true`**
- **`!`** (logical NOT, logical negation)
  - Reverses the truth/falsity of its condition
  - Unary operator, has one operand
- Useful as conditions in loops

Expression	Result
`true && false`	`false`
`true || false`	`true`
`!false`	`true`

4.5 Syntax and Logic errors

- Block:
  - Compound statements with declarations
- Syntax errors
  - Caught by compiler
- Logic errors:
  - Have their effect at execution time
  - Non-fatal: program runs, but has incorrect output
  - Fatal: program exits prematurely

4.6 Confusing Equality (==) and Assignment (=) Operators

- Dangerous error
  - Does not ordinarily cause syntax errors
  - Any expression that produces a value can be used in control structures
  - Nonzero values are **`true`**, zero values are **`false`**
  - Example using **`==`**:

```
if ( payCode == 4 )
   printf( "You get a bonus!\n" );
```

    - Checks **`paycode`**, if it is **`4`** then a bonus is awarded

Confusing Equality (==) and Assignment (=) Operators

– Example, replacing **==** with **=**:

```
        if ( payCode = 4 )
           printf( "You get a bonus!\n" );
```

- This sets **`paycode`** to **`4`**
- **`4`** is nonzero, so expression is **`true`**, and bonus awarded no matter what the **`paycode`** was

– Logic error, not a syntax error

Confusing Equality (==) and Assignment (=) Operators

- **lvalues**
  - Expressions that can appear on the left side of an equation
  - Their values can be changed, such as variable names
    - **`x = 4;`**
- **rvalues**
  - Expressions that can only appear on the right side of an equation
  - Constants, such as numbers
    - Cannot write **`4 = x;`**
    - Must write **`x = 4;`**
  - lvalues can be used as rvalues, but not vice versa
    - **`y = x;`**