IC150 Lecture 4
Decision-making

Timothy A. Gonsalves

Decisions with Variables

- Need for taking *logical decisions during problem solving*.
  - If $b^2 - 4ac$ negative, then we should report that the quadratic has no real roots.
- The *if-else* programming construct provides the facility to make logical decisions.
- Rules for usage – called *syntax* – are

if (condition) { statements if true }

else { statements if false }

Conditions

- Specified using relational and equality operators
- Relational: >, <, >=, <=
- Equality: ==, !=
- Usage: for a,b values or variables
  a > b, a < b, a >= b, a <= b, a == b, a != b.
- A condition is satisfied or true, if the relational operator, or equality is satisfied.
- For a = 3, and b = 5, a < b, a <= b, and a != b are true while a>b, a>=b, a==b are false

Completing the program

```
if (discrim < 0)
  {
     printf("no real roots, only complex\n");
     exit(1);
  }
else
  {
     root1 = (-b + sqrt(discrim))/(2*a);
     root2 = (-b - sqrt(discrim))/(2*a);
  }
```

Terminates execution and returns 1

Statements

Statement: a logical unit of instruction/command

Program: declarations and one or more statements

- assignment statement
- selection statement
- repetitive statements
- function calls etc.

All statements are terminated by semicolon (;)

Note: In C, semi-colon is a statement terminator rather than a separator!

Assignment statement

General Form:

variable " = " *expression* | *constant* ";"

The declared type of the variable should match the type of the result of expression|constant

Multiple Assignment:

var1 = *var2* = *var3* = *expression*;

var1 = (*var2* = (*var3* = *expression*));

Assignment operator associates right-to-left.

Compound Statements

A group of declarations and statements collected into a single logical unit surrounded by braces

- a block or a compound statement

"scope" of the variable declarations

- part of the program where they are applicable
- the compound statement
  - variables come into existence just after declaration
  - continue to exist till end of the block
  - unrelated to variables of the same name outside the block
  - block-structured fashion

An Example

```
{
   int i, j;
   i = 1; j =2;
   if ( expr ) {
      int i;
      i = j;
      printf("i = %d\n", i);      // output is 2
   }                              // Note: No semicolon after }
   printf("i = %d\n, i);          // output is 1
}
```

Red i and black i are different.
Bad programming style!

A compound statement can appear wherever a single statement may appear

Selection Statements

Three forms

no ***then*** reserved word

single selection:

```
if  ( att < 75 ) grade = "F";
```

double selection:

```
if (marks < 40) passed = 0; // false = 0
else passed = 1;            // true = 1
```

multiple selection:

switch statement - to be discussed later

If Statement

if (<expression>) <stmt1> [***else*** <stmt2>]

optional

Semantics:

Expression evaluates to "true"

– stmt1 will be executed

Expression evaluates to "false"

– stmt2 will be executed

Else part is optional

Expression is "true" -- stmt1 is executed

Otherwise the if statement has no effect

Grading Example

Below 50: D; 50 to 59: C ; 60 to 75: B; 75 above: A

```
int marks; char grade;
…
if (marks <= 50) grade = 'D';
else if (marks <= 59) grade = 'C';
  else if (marks <=75) grade = 'B';
    else grade = 'A';
…
```

Note the semicolon before "else"!

Unless braces are used, an else part goes with the nearest else-less if stmt

Caution in use of "else"

```
if ( marks > 40)                          /* WRONG */
  if ( marks > 75 ) printf("you got distinction");
else printf("Sorry you must repeat the course");
```

```
if ( marks > 40) {                        /*RIGHT*/
  if ( marks > 75 ) printf("you got distinction");
}
else printf("Sorry you must repeat the course");
```

Switch Statement

A multi-way decision statement

Syntax:

```
switch (expression ) {
  case const-expr : statements
  case const-expr : statements
  ...
  [ default: statements ]
}
```

[ ] ➔ optional

Counting Evens and Odds

Goal: Given several integers, count the number of even numbers and number of odd numbers

Method:

1. Initialise two counters to 0
2. While there are more numbers
   1. Read the next number
   2. Increment even count or odd count
3. Print the count of evens and odds

Sentinel <0 ➔ end

Counting Evens and Odds

```
int  num, evens = 0, odds = 0;
scanf ("%d", &num);
while (num >= 0)
  {
    switch (num%2) {
       case 0: evens++; break;
       case 1: odds++;  break;
    }
    scanf ("%d", &num);
}
printf("Even: %d, Odd: %d\n", evens, odds);
```

Counts the number of even and odd integers in the input.
Terminated by giving a negative number (*sentinel*)

Defensive programming!

Fall Throughs

Switch Statement:

Execution starts at the matching case

And *falls through* the following case statements

Unless prevented explicitly by **break** statement

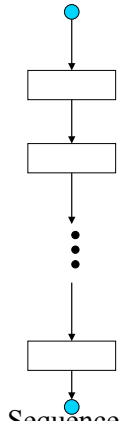
Useful for specifying one action for several cases

Break Statement:

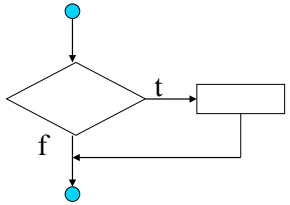
control passes to the first statement after *switch*

A feature requiring exercise of caution

Sequence and Selection Flowcharts

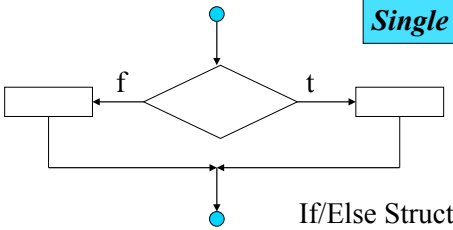


Sequence Structure



If Structure

Single Entry
Single Exit



If/Else Structure