WEEK FIVE

Control Flow

Chapter 8 Topics

- Introduction
- Selection Statements
- Iterative Statements
- Unconditional Branching
- Conclusions

Control Flow

Levels of Control Flow

- Within expressions
- Among program units
- Among program statements
  - Sequencing
  - Selection
  - Iteration

Selection Statements

- A *selection statement* provides the means of choosing between two or more paths of execution
- Two general categories:
    - Two-way selectors
    - Multiple-way selectors

Two-Way Selection Statements

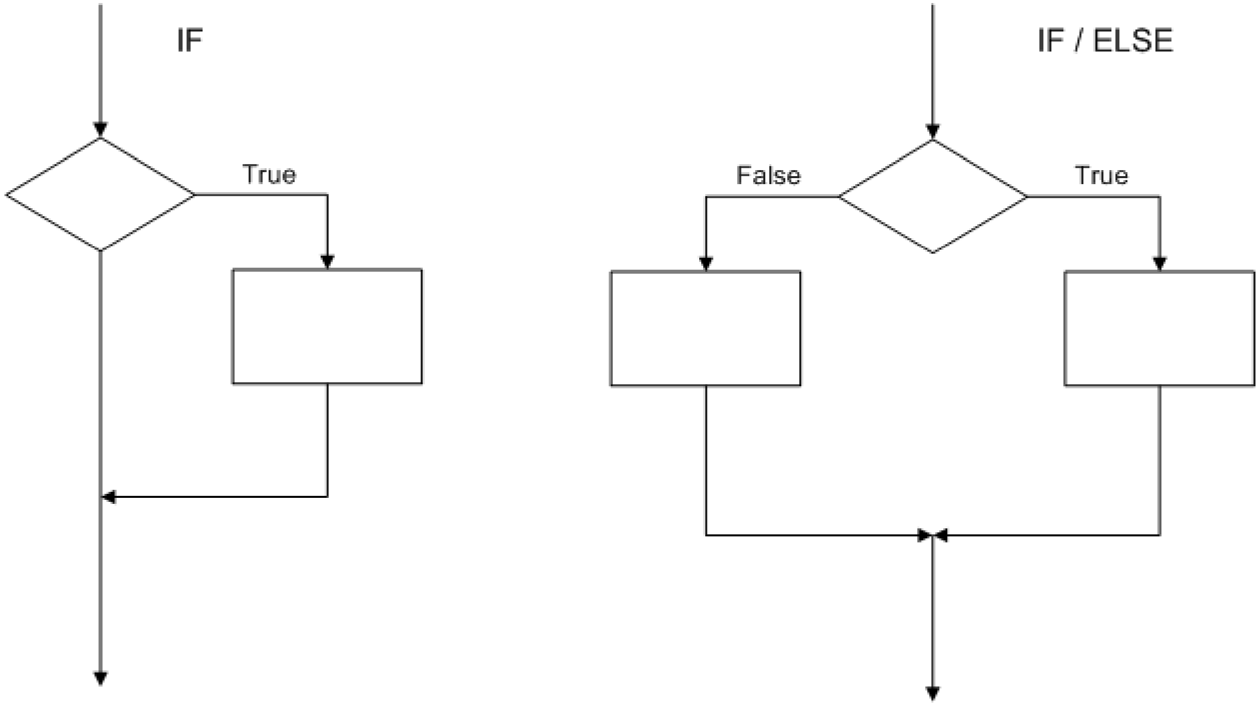
- General form:

```
if control_expression
   then clause
   else clause
```

- Design Issues:
  - What is the form and type of the control expression?
  - How are the **`then`** and **`else`** clauses specified?
  - How should the meaning of nested selectors be specified?

Two-Way Selection: C Examples

```
if (the conditional expression is true) {
     perform these statements
 }
 else {
     perform these statements if the conditional is false
 }
```



Two-Way Selection: Algol Examples

- ALGOL 60:

```
if (boolean_expr)
    then statement (then clause)
    else statement (else clause)
```

- The statements could be single or compound

Nesting Selectors

- Java example

```
 if (sum == 0)
   if (count == 0)
       result = 0;
else result = 1;
```

- Which `if` gets the `else`?
- Java's static semantics rule: `else` matches with the nearest `if`

Nesting Selectors (continued)

- To force an alternative semantics, compound statements may be used:

```
if (sum == 0) {
  if (count == 0)
      result = 0;
}
else result = 1;
```

- The above solution is used in C, C++, and C#
- Perl requires that all then and else clauses to be compound

Nesting Selectors (continued)

C/C++/Java

```
if (sum == 0) {
   if (count == 0)
     result = 0;
}
else
   result = 1;
```

Ada

```
if sum = 0 then
   if count = 0 then
     result := 0;
   end if;
else
   result := 1;
end if;
```

Multiple-Way Selection Statements

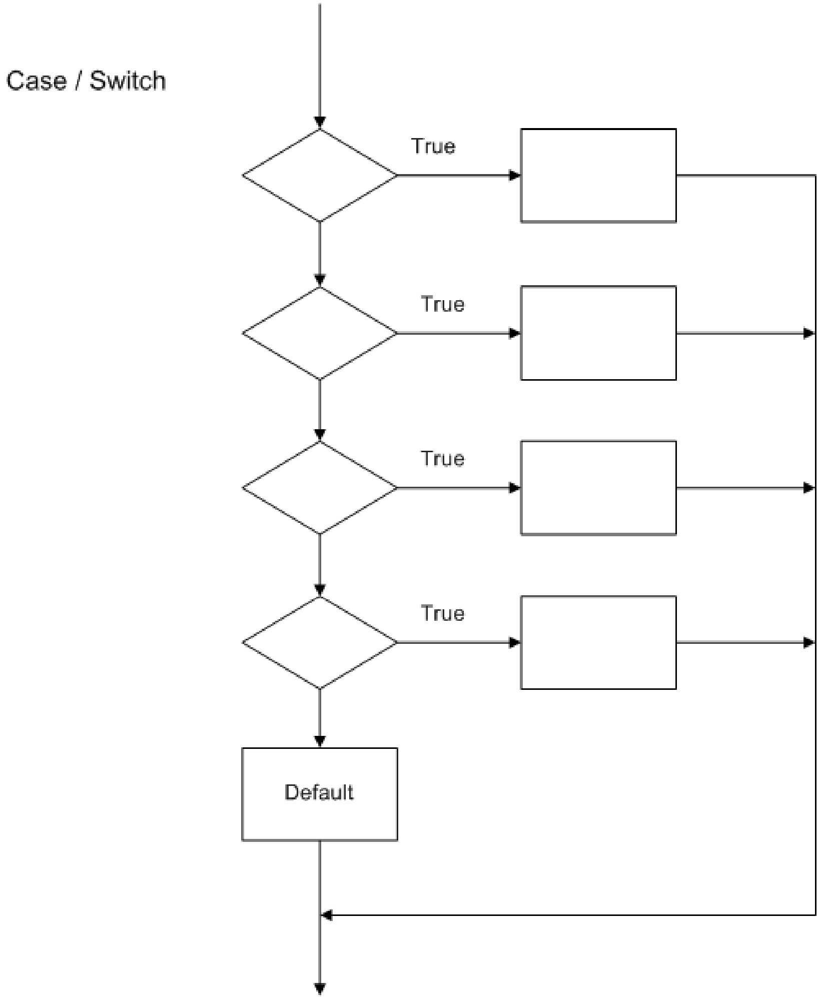
- Allow the selection of one of any number of statements or statement groups
- Design Issues:

1. What is the form and type of the control expression?
2. How are the selectable segments specified?
3. Is execution flow through the structure restricted to include just a single selectable segment?
4. What is done about unrepresented expression values?

Multiple-Way Selection: Examples

C’s switch statement

```
switch (expression) {
  case expr_1: stmt_1;
  …
  case expr_n: stmt_n;
  [default: stmt_n+1]
}
```



Multiple-Way Selection: Examples

- Design choices for C’s **`switch`** statement

1. Control expression can be only an integer type
2. Selectable segments can be statement sequences, blocks, or compound statements
3. Any number of segments can be executed in one execution of the construct (there is no implicit branch at the end of selectable segments)
4. **`default`** clause is for unrepresented values (if there is no **`default`**, the whole statement does nothing)

Multiple-Way Selection: Examples

C

```
switch (index) {
  case 1:
  case 3: odd += 1;
         sumodd += index;
         break;
  case 2:
  case 4: even +=1;
         sumeven += index;
         break;
  default:
         printf(“error”);
}
```

C#

```
switch (value) {
  case 1:
       positive++;
       break;
  case -1:
       negative++;
       break;
  case 0:
       goto case 1;
  default:
     Console.write(“error”);
}
```

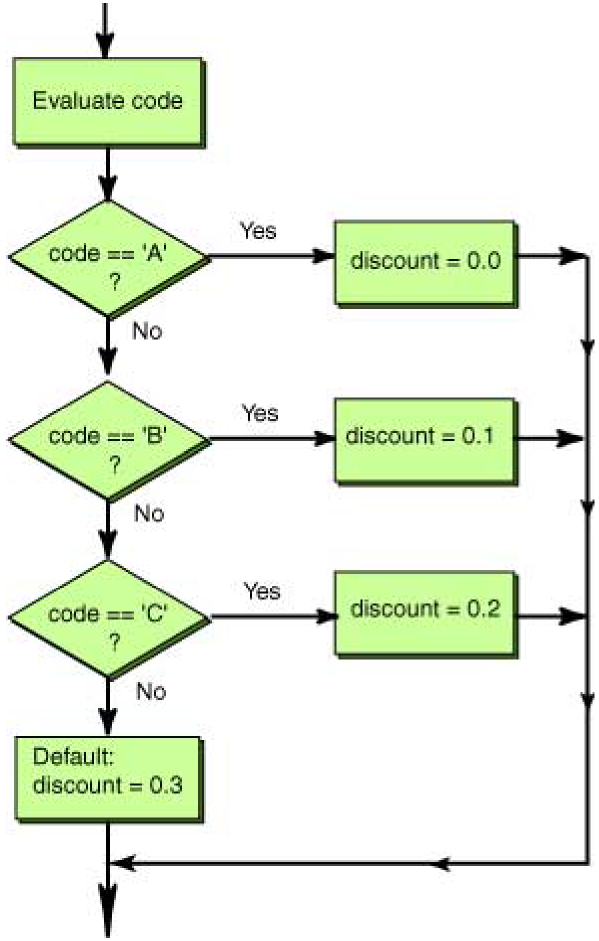
Multiple-Way Selection: Examples

- The Pascal case statement

```
CASE  ch  OF
   ‘A’,’a’: Writeln(‘ch = A’);
   ‘B’,’b’: Writeln(‘ch = B’);
   ‘C’,’c’: Writeln(‘ch = C’);
   ELSE  Writeln  (‘Try Again!’);
END;
```

More reliable than C’s `switch` (once a stmt_sequence execution is completed, control is passed to the first statement after the case statement

Multiple-Way Selection: Examples



Java

```
switch(code)
{
  case 'A':   discount = 0.0;
              break;
  case 'B':   discount = 0.1;
              break;
  case 'C':   discount = 0.2;
              break;
  default:    discount = 0.3;
}
System.out.println("discount is:
   " + discount);
```

Multiple-Way Selection Using **if**

- Multiple Selectors can appear as direct extensions to two-way selectors, using else-if clauses, for example in Ada:

```
if count<10 then
    bag1:=True;
elsif count <100 then
    bag2:=True;
elsif count <1000 then
    bag3:=True;
end if
```

```
if count<10 then
    bag1:=True;
else
  if count <100 then
      bag2:=True;
  else
    if count <1000 then
        bag3:=True;
    end if;
  end if;
end if;
```

Iterative Statements

- The repeated execution of a statement or compound statement is accomplished either by iteration or recursion
- General design issues for iteration control statements:
  1. How is iteration controlled?
  2. Where is the control mechanism in the loop?

Iterative Statements

- Two general categories:
  - Counter-Controlled Loops
  - Logically-Controlled Loops
    - Pre-test
    - Post-test
- User-Located Loop Control Mechanisms
  - Break
  - Continue
- Iteration Based on Data Structures

Counter-Controlled Loops

- A counting iterative statement has a loop variable, and a means of specifying the *initial* and *terminal*, and *stepsize* values

- Design Issues:
  1. What are the type and scope of the loop variable?
  2. What is the value of the loop variable at loop termination?
  3. Should it be legal for the loop variable or loop parameters to be changed in the loop body, and if so, does the change affect loop control?
  4. Should the loop parameters be evaluated only once, or once for every iteration?

Iterative Statements

- Pascal's for statement

```
for variable := initial (to|downto) final do
   statement
```

- Design choices:

1. Loop variable must be an ordinal type of usual scope
2. After normal termination, loop variable is undefined
3. The loop variable cannot be changed in the loop; the loop parameters can be changed, but they are evaluated just once, so it does not affect loop control
4. Just once

Iterative Statements: Examples

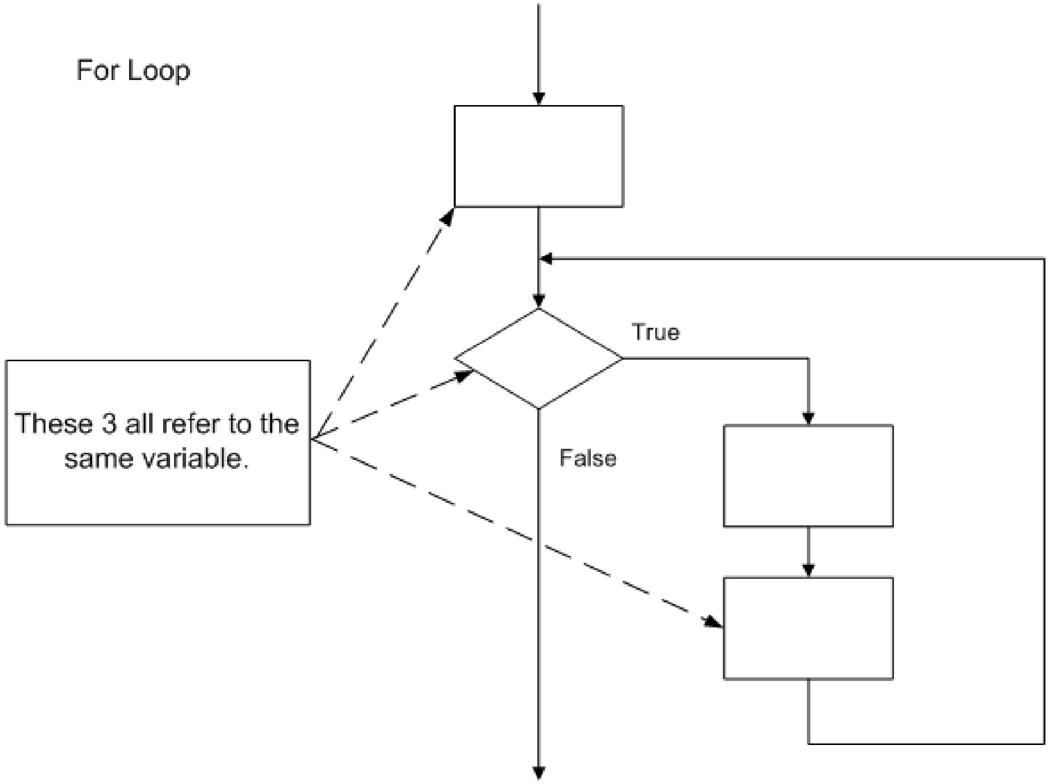
- C's `for` statement

```
for ([expr_1] ; [expr_2] ; [expr_3]) statement
```

- The expressions can be whole statements, or even statement sequences, with the statements separated by commas
  - The value of a multiple-statement expression is the value of the last statement in the expression
- There is no explicit loop variable
- Everything can be changed in the loop
- The first expression is evaluated once, but the other two are evaluated with each iteration

Iterative Statements: Examples

```
for(var initialized; conditional expression; var modifier)
{
   perform statements while the conditional expression is true
}
```



Iterative Statements: Examples

C

```
for (index = 0; index <= 10; index++)
   sum = sum + list[index];

for (count1 = 0; count2 = 1.0;
     count1 <= 10 && count2 <= 100.0;
     sum = ++count1 + count2, count2 *= 2.5);
```

C++

```
for (int count = 0; count <= len; count++)
{  ...   }
```

Iterative Statements: Examples

- C++ differs from C in two ways:
  1. The control expression can also be Boolean
  2. The initial expression can include variable definitions (scope is from the definition to the end of the loop body)
- Java and C#
  - Differs from C++ in that the control expression must be Boolean

Iterative Statements: Logically-Controlled Loops

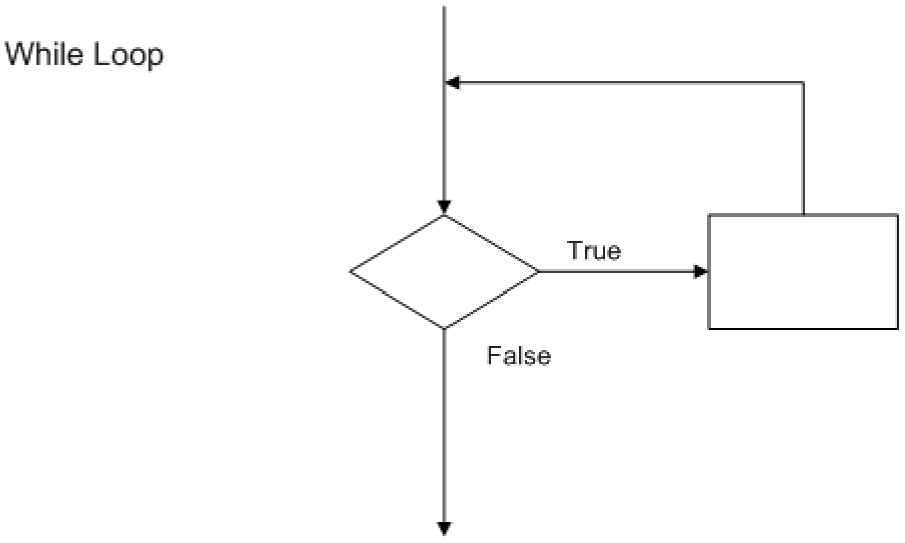
- Repetition control is based on a Boolean
- Design issues:
  - Pre-test or post-test?
  - Should the logically controlled loop be a special case of the counting loop statement ? expression rather than a counter
- General forms:

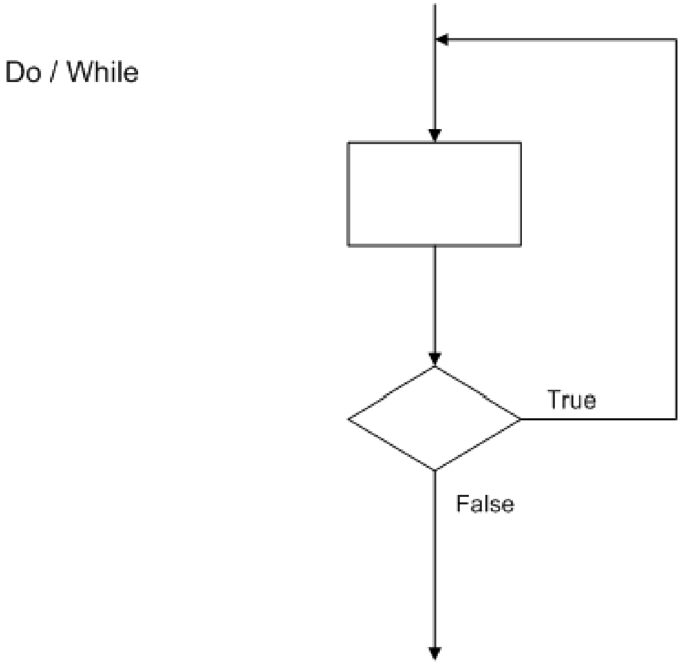
```
while (ctrl_expr)      do
  loop body              loop body
                       while (ctrl_expr)
```

Iterative Statements: Logically-Controlled Loops

```
while(conditional expr is true)
{
   perform the statements
   located between the braces
}
```

```
do {
   perform the statements
   located between the braces
}
while(conditional expr is true)
```





Iterative Statements: Logically-Controlled Loops: Examples

- Pascal has separate pre-test and post-test logical loop statements (`while-do` and `repeat-until`)
- C and C++ also have both, but the control expression for the post-test version is treated just like in the pre-test case (`while-do` and `do- while`)
- Java is like C, except the control expression must be Boolean (and the body can only be entered at the beginning -- Java has no **`goto`**

Iterative Statements: Logically-Controlled Loops: Examples

- Ada has a pretest version, but no post-test
- FORTRAN 77 and 90 have neither
- Perl has two pre-test logical loops, `while` and `until`, but no post-test logical loop

Iterative Statements: Iteration Based on Data Structures

- Number of elements of in a data structure control loop iteration
- Control mechanism is a call to an *iterator* function that returns the next element in some chosen order, if there is one; else loop is terminate
- C's **for** can be used to build a user-defined iterator:

```
for (p=root; p==NULL; traverse(p))
{ ... }
```

Iterative Statements: Iteration Based on Data Structures (continued)

- C#’s foreach statement iterates on the elements of arrays and other collections:

```
String[] strList = {“Bob”, “Carol”, “Ted”};
foreach (String name in strList)
    Console.WriteLine (“Name: {0}”, name);
```

- The notation {0} indicates the position in the string to be displayed

User-Located Control Mechanisms

- Sometimes it is convenient for the programmers to decide a location for control

Control mechanism

- break
- continue
- goto (an unconditional branching)
- return (use in function)

User-Located Control Mechanisms

break

- C , C++, and Java: **`break`** statement
- Unconditional; for any loop or **`switch`**; one level only
- Java and C# have a labeled `break` statement: control transfers to the label

continue

- An alternative: **`continue`** statement; it skips the remainder of this iteration, but does not exit the loop

User-Located Control Mechanisms: Examples

```
while (sum<1000)
{
   getnext(value);
   if (value<0)
      continue;
   sum += value;
}
```

```
while (sum<1000)
{
   getnext(value);
   if (value<0)
      break;
   sum += value;
}
```

Unconditional Branching

- Transfers execution control to a specified place in the program
- Represented one of the most heated debates in 1960's and 1970's
- Well-known mechanism: `goto` statement
- Major concern: Readability
- Some languages do not support `goto` statement (e.g., Module-2 and Java)
- C# offers `goto` statement (can be used in `switch` statements)
- Loop exit statements are restricted and somewhat camouflaged `goto`'s

Conclusion

- Variety of statement-level structures
- Choice of control statements beyond selection and logical pretest loops is a trade-off between language size and writability
- Functional and logic programming languages are quite different control structures