Chapter 8 Statement-Level Control Structure

To make programs more flexible and powerful:

– Some means of selecting among alternative control flow paths.
- Selection statement ***(conditional statements)***
- Unconditional statement

– Some means of repeating execution of certain collection of statements.
- Iterative statement

Selection Statements

- 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

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

<== which if gets the else?

- Java/C/C++/C# rule is else matches nearest if

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

- usually surrounded by braces (**{ ... }**), but some languages use **begin ... end** or other bracketing keywords.

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

- Perl requires that all then and else clauses to be compound

- Ada solution – closing special words
  – Advantage: readability

```
if ... then
   if ... then
     ...
   else
     ...
   end if
 end if
```

```
if ... then
   if ... then
     ...
   end if
 else
   ...
 end if
```

Multiple-Way Selection Statements

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

In the C-based languages:

```
if (test1) statement1
else if (test2) statement2
...
else if (testn) statementn
else default-statement
```

- The C, C++, and Java switch

```
switch ( expression )
{
   case value1 :
      statement-list1
   case value2 :
      statement-list2
   case value3 :
      statement-list3
   case  ...
   default:
}
```

switch and case are reserved

If *expression* matches *value1*, control jumps to here

Given the following switch statement where x is an int,

```
switch (x)
{
    case 3 : x = x+1;
    case 4 : x = x+2;
             break;
    case 5 : x = x+3;
    case 6 : x=x+1;
    case 7 : x = x+2;
    case 8 : x=x-1;
    case 9 : x=x+1 ;
}
```

(1) x = 4 ?
(2) x = 3 ?
(3) x = 7 ?

Put break statements after the group of statements that we want to be executed for a specific condition.

Otherwise the remainder statements will be executed until the end of the switch selective block or a break statement is reached.

• The Ada `case` statement

1. Constant lists can include
   - **Subranges e.g., 10..15**
   - **Boolean OR operators**
     for example, 1..5 | 7 | 15..20

2. Lists of constants must be exhaustive
   - **Often accomplished with `others` clause**
   - **This makes it more reliable**

```
case k is
        when 1 => j := 1;
        when 4 | 40 => j := 4;
        when 50..59 => j := 50;
        when others =>
            put ("Error in case, k =");
            put (k);
            new_line;
end case;
```

Iterative Statements

Causes a statement or collection of statements to be executed zero, or more times

– Counter controlled loops (for)
– Logically controlled loops (while, do-while)
– Iteration based on data structures (iterator)

Counter controlled loops (for)

• C Syntax **`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 first expression is evaluated once, but the other two are evaluated with each iteration.

```
for ( i = 5, j = 10 ; i + j < 20; i++, j++ )
     printf( "\n i + j = %d", (i + j) );
```

–If the second expression is absent, it is an infinite loop

• C++ and Java

–Differs from C: **The initial expression can include variable definitions (scope is from the definition to the end of the loop body)**

• Pascal's `for` statement

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

```
program next_example;
const
   MAX_STUDENTS = 100;
var
   i: integer;
   grades: array [1..MAX_STUDENTS] of char;
   num: 1..MAX_STUDENTS;
begin
   write('Enter number of students:  ');
   readln(num);
   assert((num > 0) and (num <= MAX_STUDENTS));
   for i := 1 to num do
      begin
            write('Enter grade for student ', i: 3, ':  ');
            readln(grades[i])
      end;
end;
```

Iterating over Collections and Arrays with Enhanced for

In the Java 5.0, a new kind of `for` statement was created especially for collections and arrays.

```
public class ForEachDemo {
  public static void main(String[] args) {
    int[] arrayOfInts = { 32, 87, 3, 589, 12, 1076, 2000, 8,
                          622, 127 };
    for (int element : arrayOfInts) {
        System.out.print(element + " ");
    }
  }
}
```

Iterative Statements: Logically-Controlled Loops

- General forms:

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

- Pascal has separate pre-test and post-test logical loop statements (`while-do` and `repeat-until`)
- FORTRAN 77 and 90 have neither
- Perl has pre-test logical loop, `while` and `until`, but no post-test logical loop

Loop Forever

loops forever, until an exit statement is executed within its body.

In C, it may be simulated by

```
while (true) statement
```

or

```
for (;;;) statement
```

```
while (true) {
    input := read(file);
    if (input == eof) exit;
    process(input);}
```

Loops Based on Data Types

Ada's for statement has the format

```
for variable in discrete_range loop
    statements
end loop;
```

Where `discrete_range` is subrange of an integer or enumeration type.

```
for d in Mon .. Fri loop
    printSchedule(d);
end loop;
```

Iterative Based on Data Structures

- 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** statement can be used to build a user-defined iterator:

suppose the nodes of a binary tree are to be processed. If the tree root is pointed by a variable named **root**.

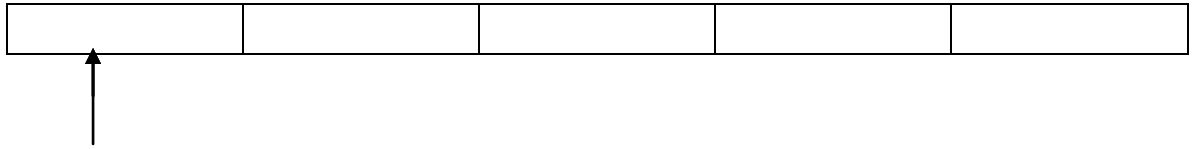
```
for (p=root; p==NULL; traverse(p)){
}
```

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

```
Strings[] strList = {"Bob", "Carol", "Ted"};
foreach (Strings name in strList)
    Console.WriteLine ("Name:", name);
```

Iterators in Java

Java supports iterator() method in most of its "container" classes – this method returns an iterator over the elements of the given collection.



iterator

- An iterator object has a **hasNext** method that returns true if there is at least one more item to process
- The **next** method returns the next item

The syntax for iterating over the elements in a Vector is as follows

```
Vector vec = new Vector;
// Populate it... Then later, iterate over its elements
Iterator it = vec.iterator ();
while (it.hasNext ()) {
  Object o = it.next ();
}
```

Unconditional Branching

- Transfers execution control to a specified place in the program
- Well-known mechanism: `goto` statement
- Major concern: Readability
- Some languages do not support `goto` statement (e.g., Module-2 and Java)
- C, C++, C# offers `goto` statement (can be used in `switch` statements)

Tutorial of C++ (part2: control structures)

Conditional structure: if and else

```
if (x > 0)
  cout << "x is positive";
else if (x < 0)
  cout << "x is negative";
else  cout << "x is 0";
```

Iteration structures (loops)

The while loop

```
#include <iostream>
using namespace std;
int main (){
  int n;
  cout << "Enter the starting number > ";
  cin >> n;
  while (n>0) {
    cout << n << ", ";
    --n;
  }
  cout << "FIRE!\n";
  return 0;}
```

```
Enter the starting number
> 8
8, 7, 6, 5, 4, 3, 2, 1,
FIRE!
```

The do-while loop

do statement while (condition);

```
#include <iostream>
using namespace std;

int main (){
  unsigned long n;
  do {
    cout << "Enter number (0 to
end): ";
    cin >> n;
    cout << "You entered: " << n
<< "\n";
  } while (n != 0);
  return 0;
}
```

```
Enter number (0 to end): 12345
You entered: 12345
Enter number (0 to end): 160277
You entered: 160277
Enter number (0 to end): 0
You entered: 0
```

The for loop

for (initialization; condition; increase) statement;

```
for ( n=0, i=100 ; n!=i ; n++, i-- )
{
   // whatever here...}
```

Jump statements.

The break statement

```
#include <iostream>
using namespace std;
int main (){
  int n;
  for (n=10; n>0; n--)  {
    cout << n << ", ";
    if (n==3)    {
      cout << "countdown
aborted!";
      break;    }
  }
  return 0;
}
```

```
10, 9, 8, 7, 6, 5, 4, 3, countdown
aborted!
```

The continue statement

The continue statement causes the program to skip the rest of the loop in the current iteration as if the end of the statement block had been reached, causing it to jump to the start of the following iteration.

```
#include <iostream>
using namespace std;

int main (){
  for (int n=10; n>0; n--) {
    if (n==5) continue;
    cout << n << ", ";
  }
  cout << "FIRE!\n";
  return 0;
}
```

```
10, 9, 8, 7, 6, 4, 3, 2, 1, FIRE!
```

The goto statement

```
 #include <iostream>
using namespace std;

int main (){
  int n=10;
  loop:
  cout << n << ", ";
  n--;
  if (n>0) goto loop;
  cout << "FIRE!\n";
  return 0;}
```

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
FIRE!
```

The exit function

Terminate the current program with a specific exit code.

```
void exit (int exitcode);
```

The exitcode is used by some operating systems and may be used by calling programs. By convention, an exit code of 0 means that the program finished normally and any other value means that some error or unexpected results happened.

The selective structure: switch.

```
switch (expression){
  case constant1:
     group of statements 1;
     break;
  case constant2:
     group of statements 2;
     break;
  .
  .
  default:
     default group of statements
}
```