HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

CSB4404 – PROGRAMMING PARADIGMS
B.Tech – VII Semester

Dr. Muthukumaran M
Associate professor
School of Computing Sciences,
Department of Computer Science and Engineering

Statement-Level Control Structures

- Introduction

- Selection Statements

- Iterative Statements

- Unconditional Branching

Introduction

ƒA control structure is a control statement and the statements whose execution it controls.

 o Selection Statements

 o Iterative Statements

 o Unconditional Branching

ƒOverall Design Question:

 o What control statements should a language have, beyond selection and pretest logical loops?

Selection Statements

ƒA selection statement provides the means of choosing between two or more paths of execution.

ƒTwo general categories:

    o Two-way selectors

    o Multiple-way selectors

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

Two-Way Selection Statements

ƒThe general form of a two-way selector is as follows:

     if control_expression

        then clause

      else clause

Design issues

- ƒWhat is the form and type of the control expression?

- ƒHow are the then and else clauses specified?

- ƒHow should the meaning of nested selectors be specified?

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

The control Expression

- ƒControl expressions are specified in parenthesis if the then reserved word is not used to introduce the then clause, as in the C-based languages.

- ƒIn C89, which did not have a Boolean data type, arithmetic expressions were used as control expressions.

- ƒIn contemporary languages, such as Java and C#, only Boolean expressions can be used for control expressions.

- Fortan includes a three-way selector named the arithmetic IF that uses an arithmetic expression for control. It causes control to go to one of three different labeled statements, depending on whether the value of its control expression is negative, zero, or greater than zero. This statement is on the obsolescent feature list of Fortran 95.

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

Clause Form

- ƒIn most contemporary languages, the then and else clauses either appear as single statements or compound statements.

- ƒC-based languages use braces to form compound statements.

- ƒIn Ada the last clause in a selection construct is terminated with end and if.

- ƒOne exception is Perl, in which all then and else clauses must be compound statements, even if they contain single statements.

Nesting Selectors

ʃIn Java and contemporary languages, the static semantics of the language specify that the else clause is always paired with the nearest unpaired then clause.

```
if (sum == 0)

        if (count == 0)

                result = 0;

else

        result = 1;
```

A rule, rather than a syntactic entity, is used to provide the disambiguation.

- So in the above example, the else clause would be the alternative to the second then clause.

- ƒTo force the alternative semantics in Java, a different syntactic form is required, in which the inner if-then is put in a compound, as in

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

ƒC, C++, and C# have the same problem as Java with selection statement nesting

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

Multiple Selection Constructs

- ƒThe multiple selection construct allows the selection of one of any number of statements or statement groups.

Design Issues

- ƒWhat is the form and type of the control expression?

- ƒHow are the selectable segments specified?

- ƒIs execution flow through the structure restricted to include just a single selectable segment?

- ƒWhat is done about unrepresented expression values?

Examples of Multiple Selectors

- *f*The C, C++, and Java switch

  switch (expression) {

  case constant_expression_1 : statement_1;

  ...

  case constant_expression_n : statement_n;

  [default: statement_n+1]

  }

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

The control expression and the constant expressions are integer type.

- ƒThe switch construct does not provide implicit branches at the end of those code segments.

- ƒSelectable segments can be statement sequences, blocks, or compound statements.

- ƒAny number of segments can be executed in one execution of the construct (a trade-off between reliability and flexibility—convenience.)

- ƒTo avoid it, the programmer must supply a break statement for each segment.

- ƒdefault clause is for unrepresented values (if there is no default, the whole statement does nothing.)

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

C# switch statement rule disallows the implicit execution of more than one segment. The rule is that every selectable segment must end with an explicit unconditional branch statement, either a break, which transfers control out of the switch construct, or a goto, which can transfer control to on of the selectable segments. C# switch statement example:

```
switch (value) {
    case -1: Negatives++;
        break;
    case 0: Positives++;
        goto case 1;
    case 1: Positives ++;
    default: Console.WriteLine("Error in switch \n");
}
```

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

Multiple Selection Using if

- ƒEarly Multiple Selectors:

    o FORTRAN arithmetic IF (a three-way selector)

        IF (arithmetic expression) N1, N2, N3

- ƒBad aspects:

    o Not encapsulated (selectable segments could be anywhere)

    o Segments require GOTOs

    o FORTRAN computed GOTO and assigned GOTO

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

To alleviate the poor readability of deeply nested two-way selectors, Ada has been extended specifically for this use.

If Count < 10 then Bag1 := True;

elsif Count < 100 then Bag2 := True;

elsif Count < 1000 then Bag3 := True;

end if;

which is equivalent to the following:
    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;

- The elsif version is the more readable of the two.

- ƒThis example is not easily simulated with a switch-case statement, because each selectable statement is chosen on the basis of a Boolean expression.

- ƒIn fact, none of the multiple selectors in contemporary languages are as general as the if-then-elsif construct.

Iterative Statement

- The repeated execution of a statement or compound statement is accomplished by iteration zero, one, or more times.

- Iteration is the very essence of the power of computer.

- The repeated execution of a statement is often accomplished in a functional language by recursion rather then by iteration.

- General design issues for iteration control statements:

    1. How is iteration controlled?

    2. Where is the control mechanism in the loop?

- The primary possibilities for iteration control are logical, counting, or a combination of the two.

- The main choices for the location of the control mechanism are the top of the loop or the bottom of the loop.

- The **body** of a loop is the collection of statements whose execution is controlled by the iteration statement.

- The term **pretest** means that the loop completion occurs before the loop body is executed.

- The term **posttest** means that the loop completion occurs after the loop body is executed.

- The iteration statement and the associated loop body together form an **iteration construct**.

Counter-Controlled Loops

- A counting iterative control statement has a var, called the **loop var**, in which the count value is maintained.

- It also includes means of specifying the **intial** and **terminal** values of the loop var, and the difference between sequential loop var values, called the **stepsize**.

- The intial, terminal and stepsize are called the **loop parameters**.

Design Issues:

– What are the type and scope of the loop variable?

– What is the value of the loop variable at loop termination?

– 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?

– Should the loop parameters be evaluated only once, or once for every iteration?

- FORTRAN 90's **DO** Syntax:

[name:]   **DO** label variable = initial, terminal [, stepsize]

        …

        END DO [name]

- The label is that of the last statement in the loop body, and the stepsize, when absent, defaults to **1**.

- Loop variable **must** be an **INTEGER** and may be either negative or positive.

- The loop params are allowed to be expressions and can have negative or positive values.

- They are evaluated at the beginning of the execution of the DO statement, and the value is used to compute an iteration count, which then has the number of times the loop is to be executed.

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

- The loop is controlled by the iteration count, not the loop param, so even if the params are changed in the loop, which is legal, those changes cannot affect loop control.

- The iteration count is an internal var that is inaccessible to the user code.

- The DO statement is a single-entry structure.

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

The for Statement of the C-Based Languages

- Syntax:

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

loop body

- The loop body can be a single statement, a compound statement, or a null statement.

for (i = 0, j = 10; j == i; i++)

…

- All of the expressions of C's for are optional.

- If the second expression is absent, it is an **infinite** loop.

- If the first and third expressions are absent, no assumptions are made.

- **The C for design choices are:**

  1. There are no explicit loop variable or loop parameters.

  2. All involved vars can be changed in the loop body.

  3. It is legal to branch into a for loop body despite the fact that it can create havoc.

- C's for is **more flexible** than the counting loop statements of Fortran and Ada, because each of the expressions can comprise multiple statements, which in turn allow multiple loop vars that can be of any type.

- Consider the following for statement:

  for (count1 = 0, count2 = 1.0;

  count1 <= 10 && count2 <= 100.0;

  sum = ++count1 + count2, count2 *= 2.5);

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

- The operational semantics description of this is:

  count1 = 0

  count2 = 1.0

  loop:

  if count1 > 10 **goto** out

  if count2 > 100.0 **goto** out

  count1 = count1 + 1

  sum = count1 + count2

  count2 = count2 * 2.5

  goto loop

  out…

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

- The loop above does not need and thus **does not** have a loop body.

- C99 and C++ differ from earlier version of C in two ways:

  1. It can use a Boolean expression for loop control.

  2. The first expression can include var definitions.

Logically Controlled Loops

- Design Issues:

1. Pretest or posttest?

2. Should this be a special case of the counting loop statement (or a separate statement)?

- C and C++ also have both, but the control expression for the posttest version is treated just like in the pretest case (**while - do** and **do - while**)

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

- These two statements forms are exemplified by the following C# code:

```csharp
sum = 0;
indat = Console.ReadLine( );
while (indat >= 0) {
        sum += indat;
        indat = Console.ReadLine( );
}
value = Console.ReadLine( );
do {
        value /= 10;
        digits ++;
} while (value > 0);
```

- The only real difference between the do and the while is that the do always causes the loop body to be executed **at least once**.

- Java does not have a **goto**, the loop bodies cannot be entered anywhere but at their beginning.

User-Located Loop Control Mechanisms

- It is sometimes convenient for a programmer to choose a location for loop control other than the top or bottom of the loop.

- **Design issues:**

  1. Should the conditional be part of the exit?

  2. Should control be transferable out of more than one loop?

- C and C++ have unconditional unlabeled exits (**break**).

- Java, Perl, and C# have unconditional labeled exits (**break** in Java and C#, **last** in Perl).

- The following is an example of nested loops in C#:

OuterLoop:

    for (row = 0; row < numRows; row++)

        for (col = 0; col < numCols; col++) {

            sum += mat[row][col];

            if (sum > 1000.0)

                break outerLoop;

}

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

- C and C++ include an unlabeled control statement, **continue**, that transfers control to the control mechanism of the smallest enclosing loop.
- This is not an exit but rather a way to skip the rest of the loop statements on the current iteration without terminating the loop structure. Ex:

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

- A negative value causes the assignment statement to be skipped, and control is transferred instead to the conditional at the top of the loop.

- On the other hand, in

while (sum < 1000) {

       getnext(value);

       if (value < 0) **break**;

       sum += value;

}

A negative value terminates the loop.

- Java, Perl, and C# have statements similar to continue, except they can include labels that specify which loop is to be continued.

- The motivation for user-located loop exits is simple: They fulfill a common need for goto statements through a highly restricted branch statement.

- The target of a goto can be many places in the program, both above and below the goto itself.

- However, the targets of user-located loop exits must be below the exit and can only follow immediately the end of a compound statement.

Unconditional Branching

An unconditional branch statement transfers execution control to a specified place in the program.

Problems with Unconditional Branching

- The unconditional branch, or **goto**, is the most powerful statement for controlling the flow of execution of a program's statements.

- However, using the goto carelessly can lead to **serious problems**.

- Without restrictions on use, imposed either by language design or programming standards, goto statements can make programs virtually unreadable, and as a result, **highly unreliable** and **difficult to maintain**.

- **C#** uses goto in the **switch** statement.

HINDUSTAN
INSTITUTE OF TECHNOLOGY & SCIENCE
(DEEMED TO BE UNIVERSITY)

- There problems follow directly from a goto's capability of forcing any program statement to follow any other in execution sequence, regardless of whether the statement proceeds or follows the first in textual order.

- Java doesn't have a goto. However, most currently popular languages include a goto statement.

CSB4404 – PROGRAMMING PARADIGMS