Principles of Programming Languages

CS F301

BITS Pilani
Dubai Campus

innovate achieve lead

BITS Pilani
Dubai Campus

Structured Programming

Topics

- Introduction
- The Need for Structured Programming
- Syntax Directed Control Flow
- Conditional statements
- Case statements
- Looping Constructs: definite and Indefinite
- Handling special cases in loop

Imperative Programming

- Imperative Programming (largely popular)
  - Variables
    - Values (state) change as programs runs
  - Computations are actions
    - Basic unit of imperative programming
    - Which occurs when programs runs
    - Change the values of variables
  - **Examples:**

1. Assignments (action) .. x = y + 2 (changes the value of x)
2. A[i] = x (changes value of array element)
3. Procedure calls like read(x) write(x) are also actions

Static Programs, Dynamic Computations


```
printf("%d %d", 1,1*1)
printf("%d %d",2,2*2)
printf("%d %d", 3,3*3)
```

```
for (i=1;i<=3;i++)
{
        printf("%d %d", i, i*i);
}
```

The program text specifies that printf(i,i*i) is executed 3 times with i taking on the value 1,2 and 3

```
while (input)
{
printf ("Hello");
}
```

Here the program text specifies printing is to be done as long as the value of input is 1

- A sequential computation consists of sequence of actions.
- Program text is static.
- Computations are dynamic that occur when program runs.
- It is essential that a **program text** represents the computation that occur when the program runs.
- **Problem:** Programmer unable to understand the actions which occur when the program runs.
- **Solution:** Design control flow constructs that are easy to understand from the program text

Design Principles of Imperative languages

1. **Structured Programming:** Structure of the program text should help us understand what program does.

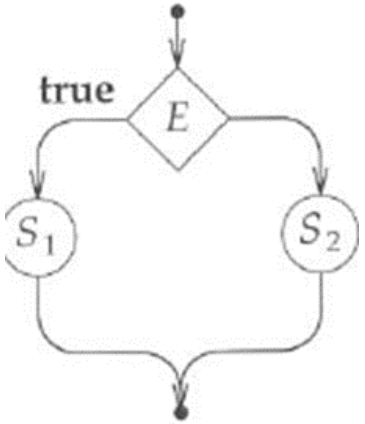
Advantage:

Readability of structured programs can make them easier for modify and tune for efficiency.

2. **Efficiency:** A language must allow an underlying machine to be used efficiently.

Syntax- Directed Control Flow.

- **Structured Programming.**
  - The structure of the (static) program text should aid us to understand what is done by the program. (readability)
- **Structured Control Flow:**
  - A program is structured if the flow of control through the program is evident from the syntactic structure of the program text.
- **How to make control flow evident from program text.**
  - By making structured statements single entry and single exit
    - Example: if E then S1 else S2
- Types of control flow statements
  1. Sequential
  2. Selection (Conditional Statements_)
  3. Iterative Construction/Loop : While/repeat.



BNF rules for statements- Syntax of C

```
stat_list           : stat
                    | stat_list stat
                    ;
selection_stat                  : 'if' '(' exp ')' stat
                    | 'if' '(' exp ')' stat 'else' stat
                    | 'switch' '(' exp ')' stat
                    ;
iteration_stat                  : 'while' '(' exp ')' stat
                    | 'do' stat 'while' '(' exp ')' ';'
                    | 'for' '(' exp ';' exp ';' exp ')' stat
                    | 'for' '(' exp ';' exp ';'     ')' stat
                    | 'for' '(' exp ';'             ';' exp ')' stat
                    | 'for' '(' exp ';'             ';'             ')' stat
                    | 'for' '('       ';' exp ';' exp ')' stat
                    | 'for' '('       ';' exp ';'     ')' stat
                    | 'for' '('       ';'             ';' exp ')' stat
                    | 'for' '('       ';'             ';'             ')' stat
                    ;
jump_stat           : 'goto' id ';'
                    | 'continue' ';'
                    | 'break' ';'
                    | 'return' exp ';'
                    | 'return'     ';'
                    ;
```

1. Sequential Statements & compound statements

Control flows sequentially through a sequence of statements.

Compound statements: Sequence of statements can be grouped into a compound statements by enclosing it between the keyword **begin** and **end.**

Eg.

temp:=x;

x:=y;

y:=temp

begin *temp* := *x*; *x* := *y*; *y* := *temp* **end**

2. Selection: Conditional Statements (if else)

Conditional Statement: selects one of the two alternative sub statements for execution.

Forms:

1. **If** <expression> **then** <statement1> **else** <statement2>
2. **If** <expression> **then** <statement>
3. Nested conditionals

If … then…

else if … then …

else if … then …

else …

2. Selection: Case Statement

- Case statements use the value of an expression to select one of the several sub-statements for execution.

```
switch (expression)
{
   case constant1:
    // statements
    break;
   case constant2:
    // statements
    break;
   case constant3:
    // statements
    break;
…….
        default:
    // default statements
}
```

- Execution begins with the evaluation of expression.
- If the value of expression = $constant_i$, control flows to the corresponding $statement_i$.

2. Selection: Case Statement

Properties

- Case constants can appear in any order.
- Case constants need not be consecutive.
- Several case constants can select the same sub statement.
- Case constants must be distinct to avoid ambiguity.

Note

- Pascal gives error if none of the cases is selected
- C allows a default case if none of the case constants are selected.

Implementation of case statement

- Case statements are used when the case constants are essentially adjacent else conditional statements can be used.
- Else part can be added to the nested conditional to achieve the effect of a default case.

```
case E of
1 :   S1;
11 :  S2;
121 : S3
end
```

(a)

```
n := E;
if n = 1 then S1
else if n = 11 then S2
else if n = 121 then S3
```

(b)

Implementation of case statement

Implementation of Case statement (at m/c code level)

- A small no. of cases is implemented using conditionals instead of case statements.
- For a large no. of cases, the range in which the case constants appear is used for creating a "jump table" array. Entry "i" in the jump table is a machine instruction that sends control to the code for case "i". *Value of the expression* is an index into the jump table, to jump to the selected case's code. If the range of case constants is min to max, then the no. of entries in the jump table = max-min+1. Only the entries for the case constants that actually appear are used.
- Compiler uses the jump table if at least half of the entries are used. If the range of case constants is large, but too many entries remain unused in the jump table, then the compiler uses the "hash table" to find the code for the selected sub-statement.

3. Looping Constructs

Divided into 2 groups depending on whether or not we can predict the number of times the loop will be executed.

1. Definite iteration:

   executed a predetermined number of times.

2. Indefinite iteration:

   The number of executions is not known when control reaches the loop

   The number is determined only at run time.

Construct 1: **while** <expression> **do** <statement>

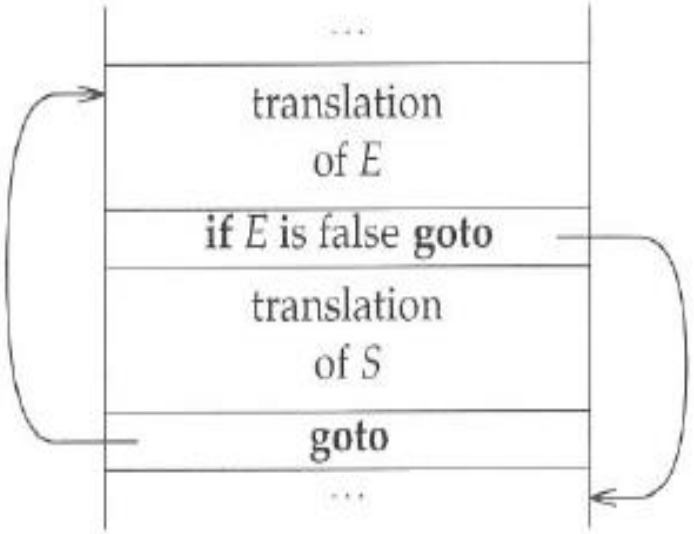
Test upon loop entry

Eg. **while** x!= 0 **do begin**… **end**

Q. What if the value of x is 0 when the control reaches the statement in Eg.

Implementation of control statement

- Implementation of while statement



3. Looping Constructs

Construct 2: **repeat** <statement-list> **until** <expression> (In Pascal)

- Allows statements to be executed repeatedly until a condition is satisfied
- Statements in the statement-list are executed before the expression condition
- If condition is not satisfied, control leaves the repeat construct else statements are repeated.

Eg. **repeat** read(next) **until** next !=0

3. Looping Constructs- for

- for <id> = < expression> to <expression> do <statement>

- For statement (in Pascal)

```
for i := 1 to limit do A[i] := 0
```

The design of for statements depends upon the treatment of:

- Index variable which controls the flow through the loop.
- Step which determines the value added to the index variable each time.
- Limit which determines when the control leaves the loop.

Handling special cases in loops

- Break and continue statements in loops:
- A break statement sends control out of the enclosing loop to the statement following the loop. Break can be used to jump out of a loop on a specified condition's satisfaction.
- A continue statement repeats the enclosing loop by sending control to the beginning of the loop.

Break statement Example in C


```
int a = 10;

/* while loop execution */
while( a < 20 ) {

   printf("value of a: %d\n", a);
   a++;

   if( a > 15) {
      /* terminate the loop using break statement */
      break;
   }
```

Output:
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15

Continue statement Example in C


```
for (int j=0; j<=8; j++)
{
    if (j==4)
    {
          /* The continue statement is encountered when
           * the value of j is equal to 4.
           */
          continue;
     }

     /* This print statement would not execute for the
      * loop iteration where j ==4  because in that case
      * this statement would be skipped.
      */
     printf("%d ", j);
}
```

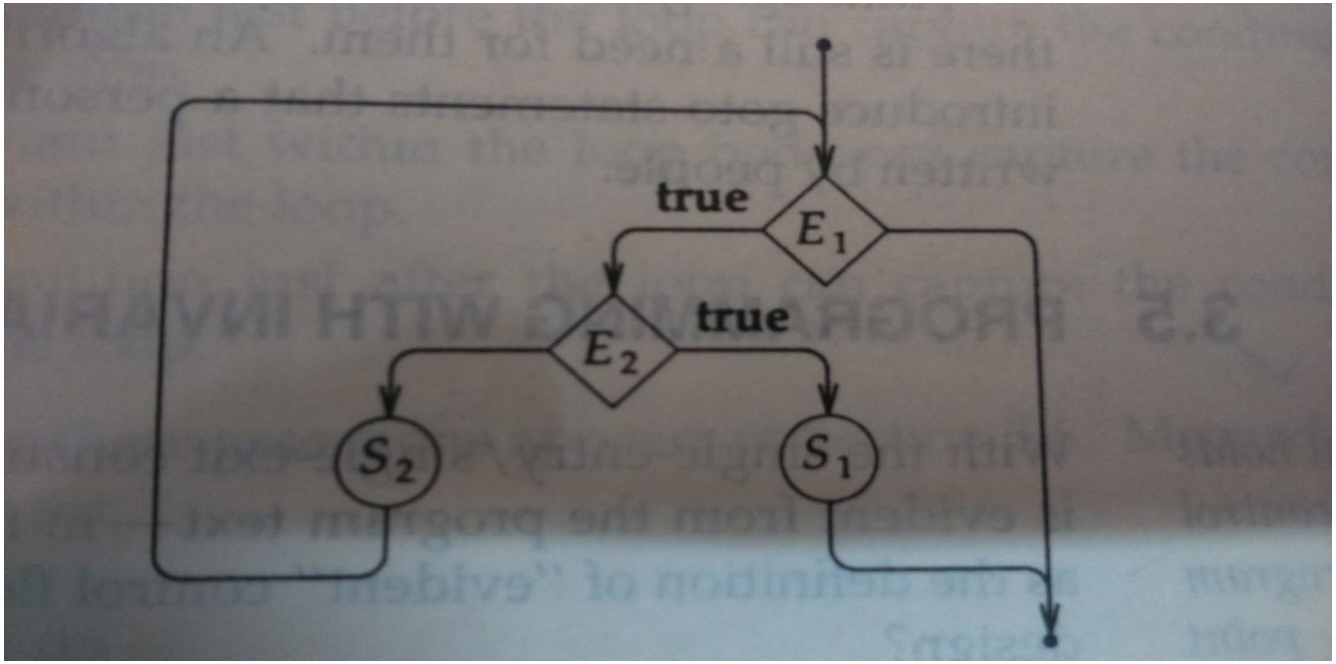
Output: 0 1 2 3 5 6 7 8

Example

```
while (E1)
{
    if (E2)
    {
        S1;
        break;
     }
    S2;
}
```

While loop has a single entry / single exit.
In this example when does the control exit out of while loop?

Flow diagram



While loop has a single entry / single exit but the control can reach the exit in 2 ways i.e. either from test E1 or after the statement S1 (due to break statement).

Return statements

return < expression>

Sends control from a procedure back to a caller carrying the value of the expression

If return is not inside a procedure the program halts

Break and return sends control out of the enclosing construct

Return : control goes out of a procedure

Break : control goes out of a loop

Example - Linear search

```
for i = n to 1 do
    if x = A[i] then
            return I;
return 0
```

Goto statements

goto L

….

….

L:<statement>

Control flows from the goto statement to a statement labeled L in the program

Control Flow in C

```
for( ; ; c = getchar() ) {
    if( c==' ' || c=='\t' )
        continue;
    if( c != '\n' )
        break;
    ++lineno;
}
```

Figure 3.17 A program fragment that uses break and continue statements.

1. For loop without assignment or condition.
2. Reads character at a time.
3. If input character equals a blank or a tab then continue statement control jumps to the beginning the loop.
4. If input character is not equal a new line then control leaves the loop through the break statement.
5. Else the line number is incremented and control goes through the loop again

Source

- Chapter 3, Ravi Sethi, "Programming Languages: Concepts and Constructs" 2nd Edition by Addison Wesley, 2006.

innovate achieve lead

BITS Pilani
Dubai Campus

Thank You!