Chapter 7: Statements

CMSC 124, 1st Semester, AY 2009-10

Chapter 7: Statements

Commands and Statements

A **command** or **statement** is a program phrase executed for the purpose of updating variables.

Kinds of Commands or Statements

1. Skips
2. Assignment statements
3. Procedure calls
4. Sequential statements
5. Collateral or concurrent statements
6. Conditional statements
7. Iterative statements

Chapter 7: Statements

Skips

- Simplest type of statement.
- Implicit after every statement.
- It does not appear explicitly but its effect is observed.

Eg 1:

```
a = b + c;
d = e * f;
```

```
a = b + c;
skip;
d = e * f;
```

Eg 2:

```
if (x > 0)
   y = y/x;
```

```
if (x > 0)
 y = y/x;
else
 skip;
```

Chapter 7: Statements

Assignment Statements

- Used to dynamically change the bindings of values to variables.
- **General Statement:**
  <target_var> <assignment_operator> <expression>
- **Assignment Operators:**

Equal Sign ("=")

- FORTRAN, BASIC, PL/I, C, C++, Java
- Creates readability problems.

Colon-Equal Sign (":=")

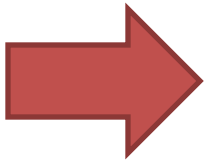
- ALGOL-60, Pascal, Modula-2, ADA
- Avoids the confusion of assignment with equality.

Chapter 7: Statements

Assignment Statements

1. **Simple Assignment**
   - Stand-alone. Destination is restricted to a single variable.

2. **Conditional Targets**
   - `flag ? count1 : count2 = 0`



```
if (flag)
   count1 = 0
else
   count2 = 0
```

3. **Multiple Statements**
   - $destination_1$:= $destination_2$:= ... := $destination_n$:= source
   - $destination_1$, $destination_2$, ..., $destination_n$:= source
   - **Eg:** `a := b := c := 0;`

Assignment Statements

4. **Simultaneous Assignment**
   - $destination_1$, $destination_2$, …, $destination_n$:= $source_1$, $source_2$, …, $source_3$
   - **Eg 1:** `a, b, c := 1, 2, 3;`
   - **Eg 2:** `a, b := b, a;`

5. **Compound Assignment (Operators)**
   - Shorthand method of specifying assignments.
   - **Eg:** `sum += value;`

6. **Unary Assignment (Operators)**
   - Another abbreviated assignment.
   - **Eg 1:** `i++;`
   - **Eg 2:** `sum = ++count;`

7. **Assignment as an Expression**
   - Assignment statement produces a result, which is the same as the value assigned to the target.
   - **Eg 1:** `while ((c = getchar()) != EOF) { … }`
   - **Eg 2:** `x = 100 + (y = 20 / z++) * 2`
   - **Eg 3:** `x = y + (y = 20 / z++) * 2` // with side effect

Chapter 7: Statements

Procedure Calls

- Change the direction of the execution.
- A procedure may return a value. In which case the procedure is called a function.
- A procedure that returns a value can be treated as an expression.
- **General Form**:

```
p(ap1, ap2, …, apn)
```

Chapter 7: Statements

Sequential Statements

- Most common control flow in imperative languages.
- **In Pascal and C:**

  $\texttt{statement}_1\texttt{; statement}_2\texttt{; statement}_3\texttt{; …, statement}_n$
- **In BASIC:**

  $\texttt{line number statement}_1\texttt{: statement}_2\texttt{: statement}_3\texttt{: …, statement}_n$

  OR

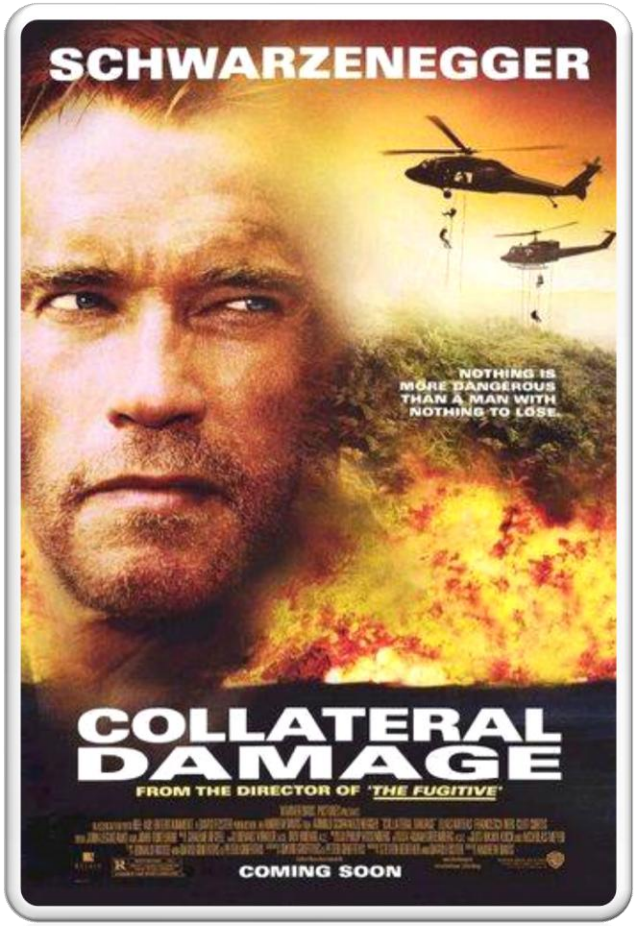
  $\texttt{line number}_1\ \texttt{statement}_1$
  $\texttt{line number}_2\ \texttt{statement}_2$
  $\texttt{line number}_3\ \texttt{statement}_3$
  …
  $\texttt{line number}_n\ \texttt{statement}_n$

Chapter 7: Statements

Collateral or Concurrent Statements

- Statements are executed in no particular order.
- They are assumed to execute simultaneously.
- **In Concurrent Pascal:**

```
cobegin
        statement1
        statement2
        ...
        statementn
coend
```



Chapter 7: Statements

Collateral or Concurrent Statements

- The n statements are executed in no particular order. This is appropriate for statements like

```
cobegin
        m := 1;
        n := n + 1;
coend
```

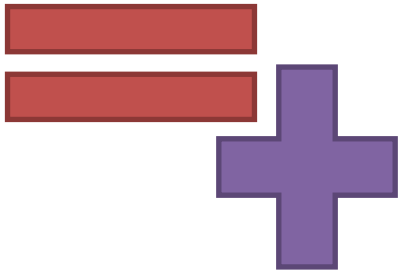
- m and n are independently updated and the order of execution is not important.

Chapter 7: Statements

Collateral or Concurrent Statements

➢It should not be used for statements like:

```
cobegin
        n := 1;
        n := n + 1;
coend
```



➢ **Possible Final Values: if n = 5**

- When **n := 1** is executed last
  ➔ 1
- When **n := n +1** is executed right after completion of n := 1
  ➔ 2
- When **n := 1** is evaluated between the evaluation of n+1 and the assignment to n
  ➔ 6

Chapter 7: Statements

Conditional Statements

- Provide the means of choosing between two or more execution paths in a program.

- **Two general categories:**
  1. Two-way selection
  2. Multiple selection

Chapter 7: Statements

Two-Way Selection

- Most conditional statements has the form:

```
if expression then statement
```

- Extension:

```
if expression then statement1 else statement2
```

- Nesting Selectors:

```
if (age > 50) then
      if (age > 75) then
            senior := true;
else
      senior := false;
```

Dangling Else

Two-Way Selection

➢ Solution in Pascal

```
if (age > 50) then
begin
        if (age > 75) then
                senior := true;
end
else
        senior := false;
```

➢ Solution in ALGOL 60

• Does not allow nesting of then with an if statement and you will get an error.

```
if (age > 50) then
begin
        if (age > 75) then
                senior := true;
end
else
        senior := false;
```

Chapter 7: Statements

Multiple Selection

- Allows the selection of one of any number of statements or statement groups.
- A generalization of a selector.
- **In ALGOL-W:**

```
case integer
expression of

begin
   statement1
   statement2
   …
   statementn
end
```

- **In Pascal:**

```
case expression of

   constantlist1:
   statement1;

   cosntantlist2:
   statement2;
    …

   constantlistn:
   statementn;
end
```

Chapter 7: Statements

Multiple Selection

- Another example **in C:**

```
switch (expression) {
    case constant1: statement1; break;
    case constant2: statement2; break;
    …
    case constantn: statementn;
}
```

The **problem** of case and switch statements is that the control variable must be an integer.

Chapter 7: Statements

Multiple Selection

- Multiple Selection **Using if:**

```
if expression1 then statement1
 elseif expression2 then statement2
 elseif expression3 then statement3
 …
 elseif expressionn then statementn
 else statement0
```

Early forms of multiple conditional statements

- **In FORTRAN:**

  `IF (expression) label1, label2, label3`

  - If negative then jump to label1
  - If 0 then jump to label2
  - If positive then jump to label3

- **Eg 1:**

```
IF (x+y) 1, 2, 3
1       …
…
GOTO 4
2       …
…
GOTO 4
3       …
…
4
```

Chapter 7: Statements

Iterative Statements

- Cause a statement or collection of statements to be executed zero, one, or more times.

- **Two types of iterative statements:**
  - Indefinite iteration
  - Definite iteration

Chapter 7: Statements

Indefinite Iteration

- This iteration is controlled by the value of a condition.

- **Sample Syntax:**

```
while expression do
    statement;
```

- **In Pascal:**

```
i := 0
while (i < 100) do
  i := i +1;
```

- **In C:**

```
i = 0;
while (i < 100)
  i++;
```

Indefinite Iteration: Post Testing

- **In C:**

```
i = -1;
do {
   i++;
} while (i < 100);
```

- **In Pascal:**

```
i := -1;
repeat
   i := i + 1;
until (i >= 100);
```

Chapter 7: Statements

Definite Iteration

- Number of iterations is known in advanced.
- Early versions of FORTRAN like FORTRAN II and IV introduced the definite iteration statement.

```
DO label variable =
initial, final [,step]
```

- The initial, final and step are integer.

- **Eg:**

```
DO 1 I = 1, 100, 1
1: SUM = SUM + I
   SUMOFSQ = SUMOFSQ
   + I * I
2:…
```

Chapter 7: Statements

Definite Iteration

- **In FORTRAN 77 and 90:**

```
DO label,  variable = initial, final [,step]
```

  - The variable was now allowed to be type integer, real, double precision.
  - Initial , final and step are allowed to be expressions.

- **In ALGOL 60:**

```
for variable := elementlist, {elementlist} do
    statement
```

where elementlist is any of the following

- expression
- expression step expression until expression
- expression while boolean_expression

Chapter 7: Statements

Definite Iteration

- **In Pascal:**
  - **Syntax:**

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

  - **Eg 1:**

```
for i := 1 to 10 do
    count := count + 1;
```

  - **Eg 2:**

```
for i := 10 downto 1 do
    count := count + 1;
```

Chapter 7: Statements

Definite Iteration

- **In ADA:**
  - **Syntax** (Similar to Pascal):

```
for variable in [reverse] range loop
   statement
end loop
```

  - **Eg 1:**

```
for i in [1..10] loop
    count := count +1;
end loop
```

  - **Eg 2:**

```
for i in reverse [1..10]
loop
    count := count +1;
end loop
```

Chapter 7: Statements

Definite Iteration

- **In Java, C, C++:**
  - **Syntax:**

```
for (expression1; expression2; expression3)
    statement;
```

  - **Eg:**

```
for (i = 1; i <=10; i++)
    count++;
```

  - You can omit any of the expression:

```
for (__;__;___)
```

Chapter 7: Statements

Definite Iteration

Three simplest forms of the statement:

/ assume count has initial value 0 */*

- ```
  for i := 1, 2, 3, 4, 5 do
      count := count + 1
  ```

- ```
  for i := 1 step 1 until 5 do
      count := count + 1;
  ```

- ```
  for i := 1, i + 1 while (i <= 5)
   do
      count := count + 1
  ```


  ```

Definite Iteration

- However, it becomes more complex when its different forms are combined. For example:

```
for i := 1, 3,
5 step 2 until 11,
3 * i while i < 100,
7, 9, 11 do
count := count + i;
```

- This code when executed will add the following values to count:
  1, 3, 5, 7, 9, 11, 33, 99, 7, 9, 11

Chapter 7: Statements

Implementation of Simple Assignment Statements

Format:

```
destination := source;
```

1. Translation of source (which is basically an expression).
2. Move the value in the accumulator to the memory address assigned to the destination.

Chapter 7: Statements

Implementation of Common Statements

✓ Statements have fixed format, implementing them is quite easy.

✓Most implementations of statements are done by the use of templates.

✓The compiler simply uses the template of the machine code equivalent of the statement.

Chapter 7: Statements

Implementation of If Statements

if expression then statement	Translation of expression If what is in the accumulator is false jump to 1: Translation of statement 1:
if expression then statement1 **else statement 2**	Translation of expression If what is in the accumulator is false jump to 1: Translation of statement1 Jump to 2: 1: Translation of statement2 2:

Chapter 7: Statements

Implementation of Loops

while expression do **statement**	1: Translation of expression If what is in the accumulator is false jump to 2: Translation of statement Jump to 1: 2:
repeat **statement** **until expression**	1: Translation of statement Translation of expression If what is in the accumulator is false jump to 1:

Chapter 7: Statements

Implementation of Loops

for control := initial to final do
statement

Translation of initial (expression)
1:
Store what is in the accumulator
to control
Translation of final (expression)
If what is in accumulator <
control jump to 2:
Translation of statement
Move value of control in the
accumulator
Add 1 to the accumulator
Jump to 1:
2:

Chapter 7: Statements

Implementation of Switch

switch (expression) {

case constant$_1$: statement$_1$;

case constant$_2$: statement$_2$;

…

case constant$_n$: statement$_n$;

}

Translation of expression
Push the value in the accumulator
    onto the stack n-1 times
Compare what is in the accumulator
    with constant1 leaving the result in
    the accumulator
If what is in the accumulator is false
    jump to 2:
Translation of statement1
Jump to exit:

Chapter 7: Statements

Implementation of Switch

2:

Pop one value from the stack and store it to the accumulator

Compare what is in the accumulator with constant2 leaving the result in the accumulator

If what is in the accumulator is false jump to 3:

Translation of statement2

Jump to exit:

n:

Pop one value from the stack and store it to the accumulator

Compare what is in the accumulator with constantn leaving the result in the accumulator

If what is in the accumulator is false jump to exit:

Translation of statementn

Jump to exit:

exit: