

## COMP205 IMPERATIVE LANGUAGES

### 11. PROGRAM CONSTRUCTS 2 (REPETITION)

- 1) Fixed count loops
- 2) Variable count loops
- 3) Post-test Loops
- 4) Terminating a loop

## REPETITION

- Two main forms of repetition:
  - 1) **Fixed Count Loops**: Repetition over a finite set (for loops).
  - 2) **Variable Count Loops**: Repetition as long as a given condition holds (while and do-while loops).
- Loops can be further classified as being either **pre-test** (while loops) or **post-test** (do-while or repeat-until loops).
- We could add recursion to the above (routine calls itself).
- Although not used so much in imperative languages recursion is the principal program construct used in logic and functional languages.

## FIXED COUNT (FOR) LOOPS

### FIXED COUNT (FOR) LOOPS

- Allow a statement to be repeated a given number of times according to a given **control variable** (i.e a counter).
  - On each loop the control value is **incremented** (or **decremented**) until the end condition is met when the loop ends.
- Issues
    - 1) Nature of the control value.
    - 2) Nature of end condition.
    - 3) Incrementation
    - 4) Decrementation
    - 5) Nesting

### NATURE OF CONTROL VARIABLE

- In some imperative languages (Ada) the control variable is initialised automatically during compilation.
- In others it must be declared specifically.

### NATURE OF END CONDITION

- On each loop the control value is tested against the end condition.
- The end condition is usually a Boolean expression or a sequence of such expressions.

### INCREMENTATION

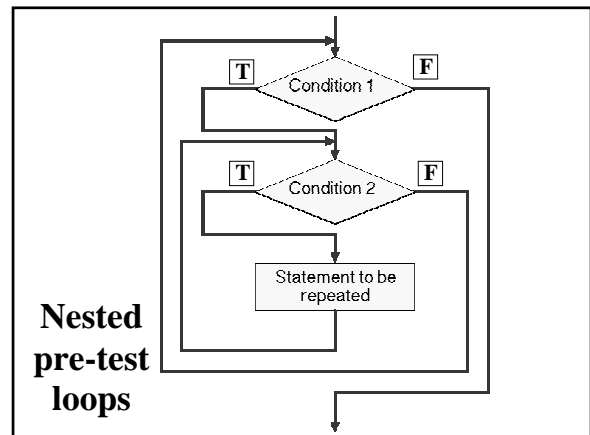
- This may be carried out automatically as designated by the compiler or as defined by the programmer.
- Incrementation is much more flexible if defined by the programmer.
- In Ada the control value is incremented automatically in steps of 1.
- In C the user defines the nature of the incrementation.

## DECREMENTATION

- In languages where the control value is automatically incremented, to process a series of statements in reverse, an adjective such as *reverse* (Ada) or *downto* (Pascal) must be included in the definition of the loop.
- In languages where the programmer controls incrementation, decrementation can be defined in the same manner.

## NESTING

- All common imperative languages support nesting of fixed count loops.



```

procedure EXAMPLE is
  DIGIT: array (integer range 0..9)
    of integer;

begin
  for INDEX in 0..9 loop
    DIGIT(INDEX) := INDEX;
  end loop;
  for INDEX in reverse 0..9 loop
    put (DIGIT(INDEX));
  end loop;
  new_line;
end EXAMPLE;

```

### ADA "FOR-LOOP" EXAMPLE (processing an array)

```

void main(void) {
  int index=0, digits[10];

  for (index;index<10;index++) {
    digits[index] = index;
  }

  for (index=9;index>=0;index--) {
    printf("%d ",digits[index]);
  }

  printf("\n");
}

```

### C "FOR-LOOP" EXAMPLE (processing an array)

```

#include<stdio.h>

void main(void) {
  int num1, num2;

  for (num1=4; num1!=1; num1--) {
    for (num2=2; num2<=6; num2=num2+2) {
      printf(" %d",num1+num2);
    }
    printf("\n");
  }
}

```

### C "NESTED FOR-LOOP" EXAMPLE

```

6 8 10
5 7 9
4 6 8

```

```

#include<stdio.h>

void main(void) {
  char letter;
  int number;

  for (letter = 'A', number = 2;
       letter < 'J'; letter++,
       number = number+5) {
    printf("%c(%d), %c(%d)\n",letter,
           letter,number,
           letter+number);
  }
}

```

### C EXAMPLE "FOR LOOP" WITH TWO CONTROL VARIABLES

```

A(65), C(67)
B(66), I(73)
C(67), O(79)
D(68), U(85)
E(69), F(91)
F(70), a(97)
G(71), g(103)
H(72), m(109)
I(73), s(115)

```

# VARIABLE COUNT (WHILE) LOOPS

## VARIABLE COUNT (WHILE) LOOPS

- *Variable count loops* (also known as *while* or *conditional loops*) allow statements to be repeated an indeterminate number of times.
- The repetition continues until a *control condition* (usually a Boolean expression) is no longer fulfilled.
- This is tested for before each repetition (*pretest* loop)
- Condition must be:
  - 1) Initialised prior to the start of the loop, and then
  - 2) Changed on each repetition of the loop.

```

procedure EXAMPLE is
  X: integer range 0..10;
begin
  put_line("Input integer 0..10")
  get(X);
  while X <= 10 loop
    put(X);
    put(X*X);
    put(X*X*X);
    new_line;
    X := X + 1;
  end loop;
end EXAMPLE;

```

```

1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
10 100 1000

```

### ADA VARIABLE COUNT (WHILE) LOOP EXAMPLE

```

void main(void) {
  int x = 1;

  printf("Input integer 0..10\n");
  scanf("%d",&x);

  while (x <= 10) {
    printf("%d %d %d\n",x,x*x,x*x*x);
    x++;
  }
}

```

### C VARIABLE COUNT (WHILE) LOOP EXAMPLE

## “FOR” LOOP OR “WHILE” LOOP?

- Both loops are *pre-test* loops.
- In languages where "for" loops can only be used to describe fixed count loops (e.g. Ada, Pascal, BASIC) the answer is straight forward:
  - For statements for fixed count loops
  - While statement for variable count loops
- In C where *for* and *while* statements can be used to describe both fixed and variable count loops, it simply a matter of style.

## COMPARISON OF C WHILE AND FOR LOOPS (Variable count examples)

```

void main(void) {
  int x;

  scanf("%d",x);

  while (x <= 10) {
    printf("%d ",x);
    printf("%d ",x*x);
    printf("%d\n",x*x*x);
    x++;
  }
}

```

```

void main(void) {
  int x;

  scanf("%d",x);

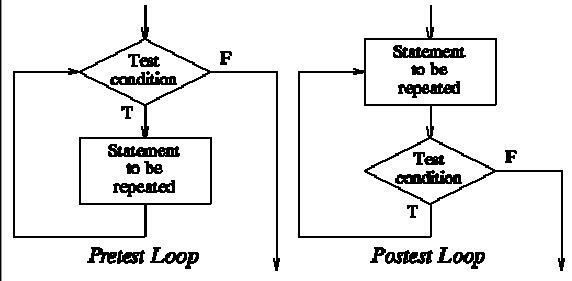
  for( ; x <= 10; x++) {
    printf("%d ",x);
    printf("%d ",x*x);
    printf("%d\n",x*x*x);
    x++;
  }
}

```

# POST-TEST LOOPS

## POST-TEST LOOPS

- Whereas for *pre-test* loops the control variable is tested prior to, the start of each iteration in *post-test* loops the control variable is tested at the end of each iteration.



```

void main(void){
int numTerms=0, frstTerm, scndTerm, tempNum;
do {
    if (numTerms == 0) {
        frstTerm = 0; printf("0");
    }
    else if (numTerms == 1) {
        scndTerm = 1; printf("1");
    }
    else {
        tempNum = frstTerm+scndTerm;
        frstTerm = scndTerm; scndTerm = tempNum;
        printf("%d ",scndTerm);
    }
    numTerms++;
} while (numTerms < 11);
printf("\n");
}
  
```

**C POSTEST DO-WHILE LOOP EXAMPLE (FIBONACCI)**

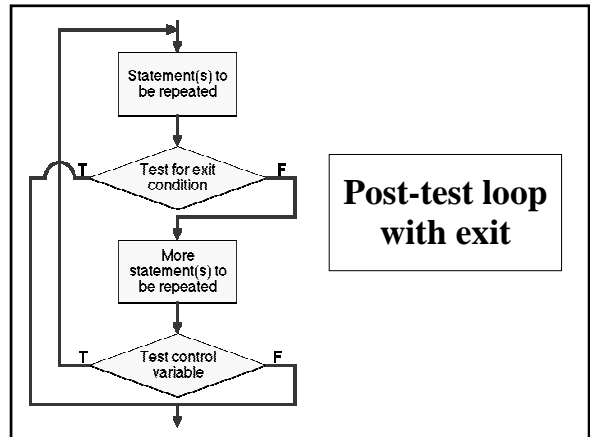
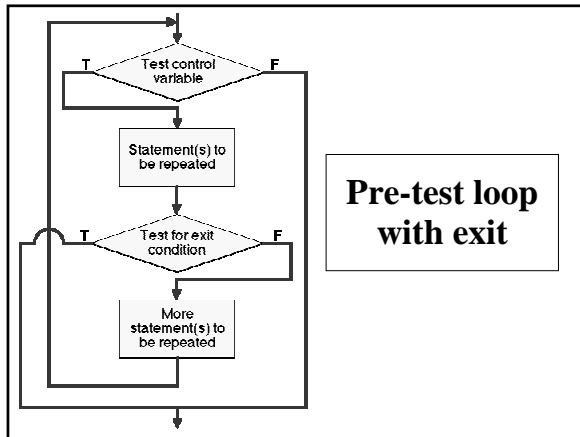
### COMPARISON OF C PRE-TEST (WHILE) AND POST-TEST (DO-WHILE) LOOPS (Input example)

<pre> void main(void) { int number = 0;  printf("Input an integer between 1 and 50 (inclusive)\n"); while (number &lt; 1    number &gt; 50) {     scanf("%d",&amp;number); } printf("%d\n",number); }   </pre>	<pre> void main(void) { int number;  printf("Input an integer between 1 and 50 (inclusive)\n"); do {     scanf("%d",&amp;number); } while (number &lt; 1    number &gt; 50); printf("%d\n",number); }   </pre>
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

# TERMINATING A LOOP

## TERMINATING A LOOP

- Sometimes there is a need to terminate a loop somewhere in the middle, for example when a *sentinel* value is reached.
- Many languages therefore supply a `break` (C) or `exit` (Ada) statement.
- Ada actually supplies two types of exit statement, `exit` and `exit when`.



```
void main(void) {
int seed;
printf("Enter seed (odd number range 1..8191)\n");
do {
scanf("%d",&seed);
if (seed < 1 || seed > 8191) {
printf("Invalid seed: %d, ",seed);
printf("seed must be range 1..8191, try again!\n");
}
else {
if (seed % 2 == 1) break;
else {
printf("Invalid seed: %d, ",seed);
printf("seed must be odd number, try again!\n");
}
} while(1);
printf("Seed = %d\n",seed);
}
```

**C LOOP EXAMPLE WITH "BREAK"**

```
procedure LOOP_EXAMPLE is
NUMBER : INTEGER ;
begin
PUT("Input an integer between ");
PUT_LINE("0 and 50 (inclusive)");
loop
GET(NUMBER);
exit when (NUMBER < 50 and NUMBER > 0);
end loop;
PUT(NUMBER);
NEW_LINE;
end LOOP_EXAMPLE;
```

**USING AN ADA "EXIT WHEN" STATEMENT TO ACHIEVE THE EFFECT OF A POST-TEST LOOP**

```
procedure LOOP_EXAMPLE is
NUMBER : INTEGER := 10;
begin
PUT("Input an integer between ");
PUT_LINE("0 and 50 (inclusive)");
while (NUMBER < 50 and NUMBER > 0) loop
GET(NUMBER);
end loop;
PUT(NUMBER);
NEW_LINE;
end LOOP_EXAMPLE;
```

**USE OF EXIT STATEMENTS CAN BE AVOIDED!**

**SUMMARY: PROGRAM CONSTRUCTS 2**

- 1) Fixed count loops
- 2) Variable count loops
- 3) Post-test Loops
- 4) Terminating a loop