

CS221: Computer Programming I

B.Sc in Computer Science

Autumn 2011

CS221: Computer Programming I

Chapter III: Control Statements

Statements to cover in this chapter

- Conditional statements
 - IF, SWITCH
- Looping statements
 - WHILE, DO ... WHILE, FOR
- Other statements
 - BREAK, CONTINUE, EXIT

Conditional statement

The **if** statement

Syntax

```
if (<expression>) <statement  
1>  
[else <statement 2>]
```

Conditional statement

The **if** statement

Description

If the `<expression>` is evaluated to non-zero (ie. True), `<statement 1>` is executed; otherwise, `<statement 2>` is executed (if it exists).

Conditional statement

The **if** statement

- Two types of if statements:
 - Simple if: if without else
 - Block if: if with else

Conditional statement

Simple if

- Valid examples (assuming appropriate declarations)
- Identify the expression and the statement part in each case.
 - `if (x < min) min = x;`
 - `if (x < 0) x = x + 1;`
 - `if (y % 2 == 0) cout << "The number is even";`
 - `If (strlen(name) > 20) cout << "Name too long";`
 - `if (grade < 50) cout << "Fail";`
 - `if (50 <= grade && grade <= 60) cout << "Good";`
 - `if (50 <= grade <= 60) cout << "Good"; // ??`
 - `if (1) cout << "Hello HiLCoE"; //??`
 - `If (!x) --x; //??`

Conditional statement

Simple if

- Invalid examples: Identify the error(s)
(assuming appropriate declarations)
 - `if x < min min = x;`
 - `if (y % 2 == 0) then cout << "The number is even";`
 - `IF (grade < 50) cout << "Not Fail";`

Conditional statement

Block if: Mutually exclusive statements

Two statements are said to be mutually exclusive if the execution of one excludes the execution of the other. **If ... else** is used when there are two mutually exclusive statements.

Conditional statement Block if

Valid example (assuming appropriate declarations)

```
if (y % 2 == 0) cout << "The number is  
even";  
else cout << "The number is odd";
```

Exercise

Write a program that receives a number from a keyboard and tells whether a number is negative, zero or positive.

Is this solution correct?

```
void main()
{
    int x;
    cout << "Enter a number > ";
    cin >> x;
    if (x < 0) cout << "Negative";
    if (x == 0) cout << "Zero";
    if (x > 0) cout << "Positive";
}
```

What about this one?

```
void main()
{
    int x;
    cout << "Enter a number > ";
    cin >> x;
    if (x < 0) cout << "Negative";
    if (x == 0) cout << "Zero";
    else cout << "Positive";
}
```

What about this one?

```
void main()
{
    int x;
    cout << "Enter a number > ";
    cin >> x;
    if (x < 0) cout << "Negative";
    else if (x == 0) cout << "Zero";
    else (x > 0) cout << "Positive";
}
```

What about this one?

```
void main()
{
    int x;
    cout << "Enter a number > ";
    cin >> x;
    if (x < 0) cout << "Negative";
    else if (x = 0) cout << "Zero";
    else cout << "Positive";
}
```

Best solution

```
void main()
{
    int x;
    cout << "Enter a number > ";
    cin >> x;
    if (x < 0) cout << "Negative";
    else if (x == 0) cout << "Zero";
    else cout << "Positive";
}
```

Decompose the if - else statement

```
void main()
{
    int x;
    cout << "Enter a number > ";
    cin >> x;
    if (x < 0) cout << "Negative";
    else if (x == 0) cout << "Zero";
    else cout << "Positive";
}
```

Decompose the if - else statement

```
void main()
{
    int x;
    cout << "Enter a number > ";
    cin >> x;
    if (x < 0) cout << "Negative";
    else if (x == 0) cout << "Zero";
    else cout << "Positive";
}
```

Decompose the if - else statement

```
void main()
{
    int x;
    cout << "Enter a number > ";
    cin >> x;
    if (x < 0) cout << "Negative";
    else if (x == 0) cout << "Zero";
    else cout << "Positive";
}
```

Income tax calculation

```
void main()
{ float salary, tax;
  cout << "Enter monthly salary: ";
  cin >> salary;
  if (salary <= 150) tax = 0;
  else if (151<= salary && salary <=650) tax = (salary - 150)*0.1;
  else if (651<= salary && salary <=1400) tax = 50 + (salary -
    650)*0.15;
  else if (1401<= salary && salary <=2350) tax = 162.50+(salary -
    1400)*0.2;
  else if (2351<= salary && salary <=3550)
    tax = 352.50+(salary - 2350)*0.25;
  else if (3551<= salary && salary <=5000)
    tax = 652.50+(salary - 3550)*0.3;
  else tax = 1087.50 + (salary - 5000)*0.35;
  cout << "Income tax = " << tax << endl;
  cout << "Net Income = " << salary - tax << endl;
}
```

Conditional statement

The **switch** statement

Left as an exercise.

Exercise

Write a program that accepts two integers and a binary operator (+, -, *, /) and displays the result of applying the operator on the numbers.
(use the switch statement)

Anything wrong with this solution?

```
void main()
{ int x, y;
  char opr;
  cout << "Enter two numbers: ";
  cin >> x >> y;
  cout << "Enter a binary operator [ +, -, *, / ] >";
  cin >> opr;
  cout << "The result is " << x opr y << endl;
}
```

What about in this?

```
void main()
{ int x, y;
  char opr;
  cout << "Enter two numbers: ";
  cin >> x >> y;
  cout << "Enter a binary operator [ +, -, *, / ] > ";
  cin >> opr;
  switch (opr) {
    case '+': { cout << "Addition = " << x + y; break;}
    case '-': { cout << "Subtraction = " << x - y; break;}
    case '*': { cout << "Multiplication = " << x * y; break;}
    case '/': { cout << "Division = " << x / y; break;}
    default:
      cout << "Invalid operator" << endl;
  }
}
```

What about in this?

```
void main()
{ int x, y;
  char opr;
  cout << "Enter two numbers: ";
  cin >> x >> y;
  cout << "Enter a binary operator [ +, -, *, / ] > ";
  cin >> opr;
  switch (opr) {
    case '+': { cout << "Addition = "      << x + y; break;}
    case '-': { cout << "Subtraction = "   << x - y; break;}
    case '*': { cout << "Multiplication = " << x * y; break;}
    case '/': { if (y!=0) cout << "Division = " << x / y;
                else cout << "Division: Denominator can not be 0";
                break;
              }
    default:
      cout << "Invalid operator" << endl;
  }
}
```

Looping Statements

- While
- Do while
- For

The while statement

Syntax

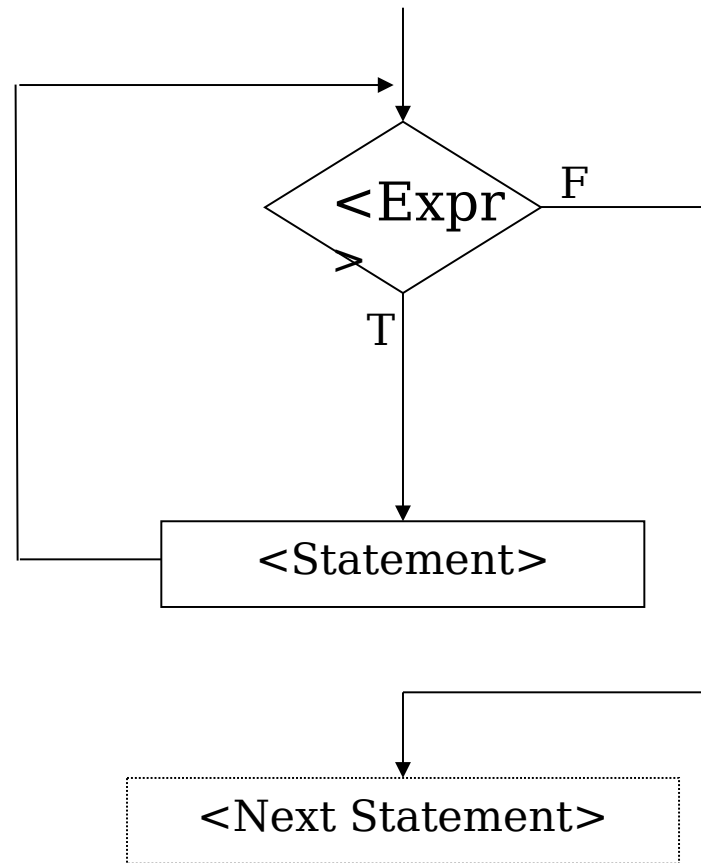
```
while (<expression>)  
<statement>
```

The while statement

Description

Repeatedly execute the `<statement>` as long as the `<expression>` is evaluated to non-zero (ie. True).

Schematic description of while



Example of valid while statements

- (assuming appropriate declarations)
- `while (n >=0) {`
 - `cout << "Hello HiLCoE";`
 - `n--;`
 - `}`
- `while (i<=n) {`
 - `cout << "Enter next data > ";`
 - `cin >> x;`
 - `If (x < min) min = x;`
 - `i++;`
- `}`

Also valid ...: What is accomplished?

- `while (1) cout << "Hello HiLCoE";`
- `while (0) cout << "Hello HiLCoE";`
- `int i=1;`
- `while (i<=10) cout << "Hello
HiLCoE";`
 - `i++;`

While Exercise

Write a program the counts for the number of -ves, zeros and +ves in a list of numbers.

>= two days absence: to report to office

- Daniel Solomon
- Tewodros Seifemichael
- Alexander Abraham
- Yonas Asfaw

Solution to the exercise

- Identify the repeating task (RT)
- Identify the terminating (or continuing) condition (TC)

Repeating task

- Receiving a number
- Checking whether the number is -ve, zero or +ve
- Incrementing the counting by one

Terminating condition: Two ways

1. As long as the number of elements entered is not yet equal to the size of the list.
2. Until a special element which is not an element of the list (called sentinel) is entered.

Terminating condition: using the size of the list

1. As long as the number of elements entered is not yet equal to the size of the list:
 - Need to enter the size of the list before starting to enter the elements of the list.
 - Need to count the elements already entered and processed for each element of the list.
 - Need to compare the number of elements processed and the size of the list each time an element is entered and processed.

Terminating condition: using sentinel

2. Until a special element which is not an element of the list (called sentinel) is entered:
 - Need to enter the sentinel before starting to enter the elements of the list.
 - Need to compare each element with the sentinel as it is entered from the keyboard and before it is processed.

Are you ok with this solution (1)?

```
• int main()
• { int n, i, nneg, nzero, npos;
•   float x;
•   cout << "Enter the size of the list > ";
•   cin >> n;
•   i=1;
•   while (i<=n) {
•     • cout << "Enter an element of the list > ";
•     • cin >> x;
•     • if (x < 0) nneg++;
•     • else if (x == 0) nzero++;
•     • else npos++;
•     • i++;
•   }
•   cout << "Number of -ve numbers = " << nneg << endl;
•   cout << "Number of zeros = " << nzero << endl;
•   cout << "Number of +ve numbers = " << npos << endl;
•   return 0;
• }
```

And what about this one (2)?

```
• int main()
• { int n, i, nneg, nzero, npos;
•   float x;
•   cout << "Enter the size of the list > ";
•   cin >> n;
•   nneg = nzero = npos = 0;
•   while (i<=n) {
•     • cout << "Enter an element of the list > ";
•     • cin >> x;
•     • if (x < 0) nneg++;
•     • else if (x == 0) nzero++;
•     • else npos++;
•   }
•   cout << "Number of -ve numbers = " << nneg << endl;
•   cout << "Number of zeros = " << nzero << endl;
•   cout << "Number of +ve numbers = " << npos << endl;
•   return 0;
• }
```

And this one (3)?

```
int main()
{
    int n, i, nneg, nzero, npos;
    float x;
    cout << "Enter the size of the list > ";
    cin >> n;
    nneg = nzero = npos = 0;
    i = 0;
    while (i<=n) {
        cout << "Enter an element of the list > ";
        cin >> x;
        if (x < 0) nneg++;
        else if (x == 0) nzero++;
        else npos++;
    }
    cout << "Number of -ve numbers = " << nneg << endl;
    cout << "Number of zeros = " << nzero << endl;
    cout << "Number of +ve numbers = " << npos << endl;
    return 0;
}
```

A correct solution

```
int main()
{
    int n, i, nneg, nzero, npos;
    float x;
    cout << "Enter the size of the list > ";
    cin >> n;
    nneg = nzero = npos = 0;
    i = 1;
    while (i<=n) {
        cout << "Enter an element of the list > ";
        cin >> x;
        if (x < 0) nneg++;
        else if (x == 0) nzero++;
        else npos++;
        i++;
    }
    cout << "Number of -ve numbers = " << nneg << endl;
    cout << "Number of zeros = " << nzero << endl;
    cout << "Number of +ve numbers = " << npos << endl;
    return 0;
}
```

A solution with sentinel: Is this ok (1)?

```
int main()
{
    int nneg, nzero, npos;
    float x, s;
    cout << "A sentinel element is the last element to be
    entered to indicate the termination of entering the elements
    of the list \n";
    cout << "Enter a number that will serve as sentinel > ";
    cin >> s;
    nneg = nzero = npos = 0;
    while (x != s) {
        cout << "Enter an element of the list > ";
        cin >> x;
        if (x < 0) nneg++;
        else if (x == 0) nzero++;
        else npos++;
    }
    cout << "Number of -ve numbers = " << nneg << endl;
    cout << "Number of zeros = " << nzero << endl;
    cout << "Number of +ve numbers = " << npos << endl;
    return 0;
}
```

A solution with sentinel: And this one (2)?

```
int main()
{ int nneg, nzero, npos;
  float x, s;
  cout << "A sentinel element is the last element to be
    entered to indicate the termination of entering the elements
    of the list \n";
  cout << "Enter a number that will serve as sentinel > ";
  cin >> s;
  nneg = nzero = npos = 0;
  cout << "Enter an element of the list > ";
  cin >> x;
  while (x != s) {
    if (x < 0) nneg++;
    else if (x == 0) nzero++;
    else npos++;
  }
  cout << "Number of -ve numbers = " << nneg << endl;
  cout << "Number of zeros = " << nzero << endl;
  cout << "Number of +ve numbers = " << npos << endl;
  return 0;
}
```

A solution with sentinel: And this one (3)?

```
int main()
{
    int nneg, nzero, npos;
    float x, s;

    cout << "A sentinel element is the last element to be entered to
    indicate the termination of entering the elements of the list \n";
    cout << "Enter a number that will serve as sentinel > ";
    cin >> s;
    nneg = nzero = npos = 0;
    cout << "Enter first element of the list > ";
    cin >> x;
    while (x != s) {
        if (x < 0) nneg++;
        else if (x == 0) nzero++;
        else npos++;
        cout << "Enter the next element of the list > ";
        cin >> x;
    }
    cout << "Number of -ve numbers = " << nneg << endl;
    cout << "Number of zeros = " << nzero << endl;
    cout << "Number of +ve numbers = " << npos << endl;
    return 0;
}
```

Exercise

Write a program that identifies the smallest digit in a given positive integer number.

Searching for the smallest digit

- void main()
- { unsigned long num;
- unsigned int minDig, rd;
- cout << "Enter a number > ";
- cin >> num;
- minDig = 9;
- while (num > 0) {
- rd = num % 10;
- if (rd < minDig) minDig = rd;
- num = num / 10;
- }
- cout << "the minimum digit is " << minDig;
- }

Counts for the occurrences of the minimum digit

```
void main()
{ unsigned long num;
  unsigned int minDig, rd, minCount;
  cout << "Enter a number > ";
  cin >> num;
  minCount = 0;
  minDig = 9;
  while ( num > 0) {
    rd = num % 10;
    if (rd == minDig) minCount++;
    else if (rd < minDig) {
      minDig = rd;
      minCount = 1;
    }
    num = num / 10;
  }
  cout << "the minimum digit is " << minDig << endl;
  cout << " and it occurs " << minCount << " times ";
}
```

Identifying telephone codes (1)

- `void main( )`
- `{ const stdNbDigits = 9;`
 `/* Standard number of digits in Ethn telephone system without`
 `counting 0 */`
- `unsigned long telephoneNum, leftTwoDigits, actualNbDigits,`
 `tempo;`
- `clrscr();`
- `cout << "\n\n Enter a telephone number [without the leading 0] >`
 `";`
- `cin >> telephoneNum;`
- `// check if the telephoneNum has the required number of digits`
- `tempo = telephoneNum;`
- `actualNbDigits=0; while (tempo > 0) {tempo = tempo /10;`
 `actualNbDigits++;}`
- `if (actualNbDigits == stdNbDigits) {`
 `leftTwoDigits = telephoneNum / pow(10, stdNbDigits - 2);`

Identifying telephone codes (2)

```
cout << "\n\n Tel = " << telephoneNum << endl;
    switch(leftTwoDigits) {
        case 91: cout << "\n\n The number is mobile \n "; break;
        case 11: cout << "\n\n The number is Addis Ababa \n "; break;
        case 22: cout << "\n\n The number is South Eastern Region (Adama) \n "; break;
        case 25: cout << "\n\n The number is Eastern Region (DireDawa) \n "; break;
        case 33: cout << "\n\n The number is North Eastern Region (Dessie) \n "; break;
        case 34: cout << "\n\n The number is Northern Region (Mekelle) \n "; break;
        case 46: cout << "\n\n The number is Southern Region (Hawassa) \n "; break;
        case 47: cout << "\n\n The number is South Western Region (Jimma) \n "; break;
        case 57: cout << "\n\n The number is Western Region (Nekempt) \n "; break;
        case 58: cout << "\n\n The number is North Western Region (BahrDar) \n ";
        break;
        default: cout << "\n\n This number is a wrong telephone number!!" << endl;
    }
}
else cout << "\n\n\n\t\t This is not a standard ETC number!! ";
}
```

The do ... while statement

Syntax

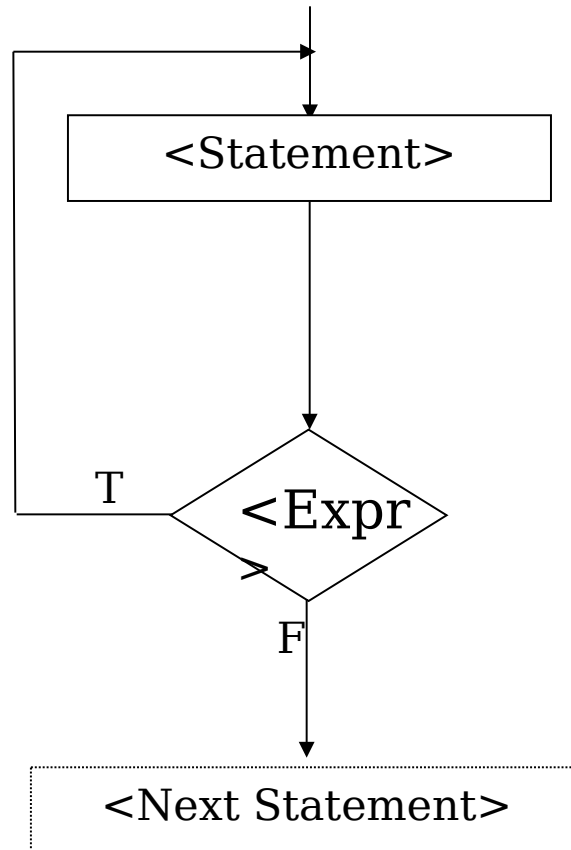
```
do <statement> while  
(<expression>);
```

The do ... while statement

Description

The <statement> is executed once and then it is repeatedly executed as long as the <expression> is evaluated to non-zero (ie. True).

Schematic description of do while



Example of valid do while statements

- (assuming appropriate declarations)
- do {
 - cout << "Hello HiLCoE";
 - n--;
- }
- while (n >=0);
- do {
 - cout << "Enter next data > ";
 - cin >> x;
 - If (x < min) min = x;
 - i++;
- }
- while (i<=n);

Also valid ...: What is accomplished?

- `do cout << "Hello HiLCoE"; while (1);`
- `do cout << "Hello HiLCoE"; while (0);`
- `int i=1;`
- `do cout << "Hello HiLCoE"; while (i<=10);`
 - `i++;`

Do ... While

Do ... While is more appropriate when the repeating task is executed at least once.

Exercise

Write a program that converts a decimal number into binary.

Decimal into binary: Is this solution right?

- `void main()`
- `{ unsigned short num;`
 - `cout << "enter a positive number > ";`
 - `cin >> num;`
 - `do {`
 - `cout << num % 2;`
 - `num = num / 2;`
 - `} while (num > 0);`
- `}`

Exercise

Write a program that plays
“game of fork”.

Game of fork: Any problem?

- `void main()`
- `{ unsigned char hidden, guess;`
- `hidden = 12;`
 - `cout << "I chose a number between 1 & 50 \n";`
 - `cout << "Up to you to guess it ... \n";`
 - `cout << "Enter your guess > ";`
 - `do {`
 - `cin >> guess;`
 - `If (hidden != guess) cout << "Try again ... > ";`
 - `} while (hidden != guess);`
- `cout << "Bravo - you got it!";`
- `}`

Game of fork: improved(1)

- `void main()`
- `{ unsigned char hidden, guess;`
- `randomize();`
- `hidden = random(50) + 1;`
 - `cout << "I chose a number between 1 & 50 \n";`
 - `cout << "Up to you to guess it ... \n";`
 - `cout << "Enter your guess > ";`
 - `do {`
 - `cin >> guess;`
 - `If (hidden != guess) cout << "Try again ... > ";`
 - `} while (hidden != guess);`
- `cout << "Bravo - you got it!";`
- `}`

Game of fork: improved(2)

- void main()
- { unsigned char hidden, guess;
- randomize();
- hidden = random(50) + 1;
 - cout << "I chose a number between 1 & 50 \n";
 - cout << "Up to you to guess it ... \n";
 - cout << "Enter your guess > ";
 - do {
 - cin >> guess;
 - If (guess < hidden) cout << "Try a bigger number ... > ";
 - else if (guess > hidden) cout << "Try a smaller number ...> ";
 - Else cout << "Bravo - you got it!";
 - } while (hidden != guess);
- }

Game of fork: better still (3)

- `void main()`
- `{ unsigned char hidden, guess;`
- `randomize();`
- `hidden = random(50) + 1;`
 - `cout << "I chose a number between 1 & 50 \n";`
 - `cout << "Up to you to guess it ... \n";`
 - `cout << "Enter your guess > ";`
 - `do {`
 - `cin >> guess;`
 - `If (guess < hidden) cout << "Try a bigger number ... > ";`
 - `else if (guess > hidden) cout << "Try a smaller number ...> ";`
 - `} while (hidden != guess);`
 - `cout << "Bravo - you got it!";`
- `}`

Game of fork: Limiting the number of trials – is the following code ok (4) ?

- `void main()`
- `{ const MAX = 5;`
- `unsigned char hidden, guess, steps;`
- `randomize();`
- `hidden = random(50) + 1;`
 - `cout << "I chose a number between 1 & 50 \n";`
 - `cout << "Up to you to guess it in a maximum of " << MAX << " steps \n";`
 - `cout << "Enter your guess > ";`
 - `steps = 1;`
 - `do {`
 - `cin >> guess;`
 - `If (guess < hidden) cout << "Try a bigger number ... > ";`
 - `else if (guess > hidden) cout << "Try a smaller number ...> ";`
 - `steps++;`
 - `} while (hidden != guess && steps <= MAX);`
 - `cout << "Bravo - you got it!";`
- `}`

Game of fork: Limiting the number of trials – and this one (5) ?

```
• void main()
• { const MAX = 5;
•   unsigned char hidden, guess, steps;
•   randomize();
•   hidden = random(50) + 1;
•   cout << "I chose a number between 1 & 50 \n";
•   cout << "Up to you to guess it in a maximum of " << MAX << " steps \n";
•   cout << "Enter your guess > ";
•   steps = 1;
•   do {
•     cin >> guess;
•     If (guess < hidden) cout << "Try a bigger number ... > ";
•     else if (guess > hidden) cout << "Try a smaller number ...> ";
•     steps++;
•   } while (hidden != guess && steps <= MAX);
•   If (hidden == guess) cout << "Bravo - you got it!";
•   else cout << "Sorry, you finished your chances";
• }
```

The for statement

Syntax

```
for ([<exp1>]; [<exp2>];  
[<exp3>])  
    <statement>
```

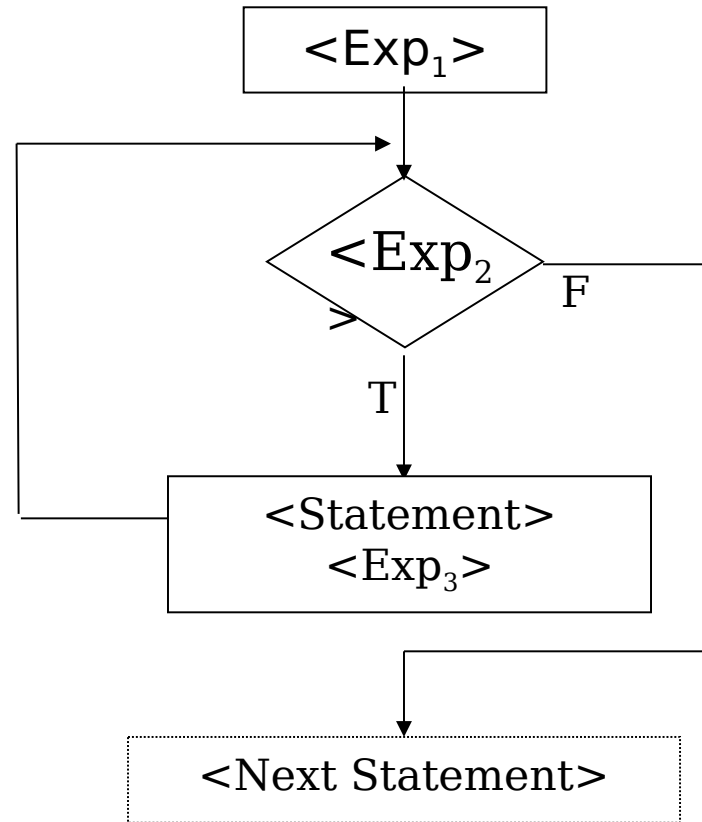
The for statement

$\langle \text{exp}_1 \rangle$: Initialization expression

$\langle \text{exp}_2 \rangle$: Terminating condition

$\langle \text{exp}_3 \rangle$: Inc/dec expression

Schematic description of for



Example of valid for statements

- (assuming appropriate declarations)
- for (n=10; n>=0; n--)
 - cout << "Hello HiLCoE"
- for (i=0; i<=n; i++) {
 - cout << "Enter next data > ";
 - cin >> x;
 - If (x < min) min = x;
- }

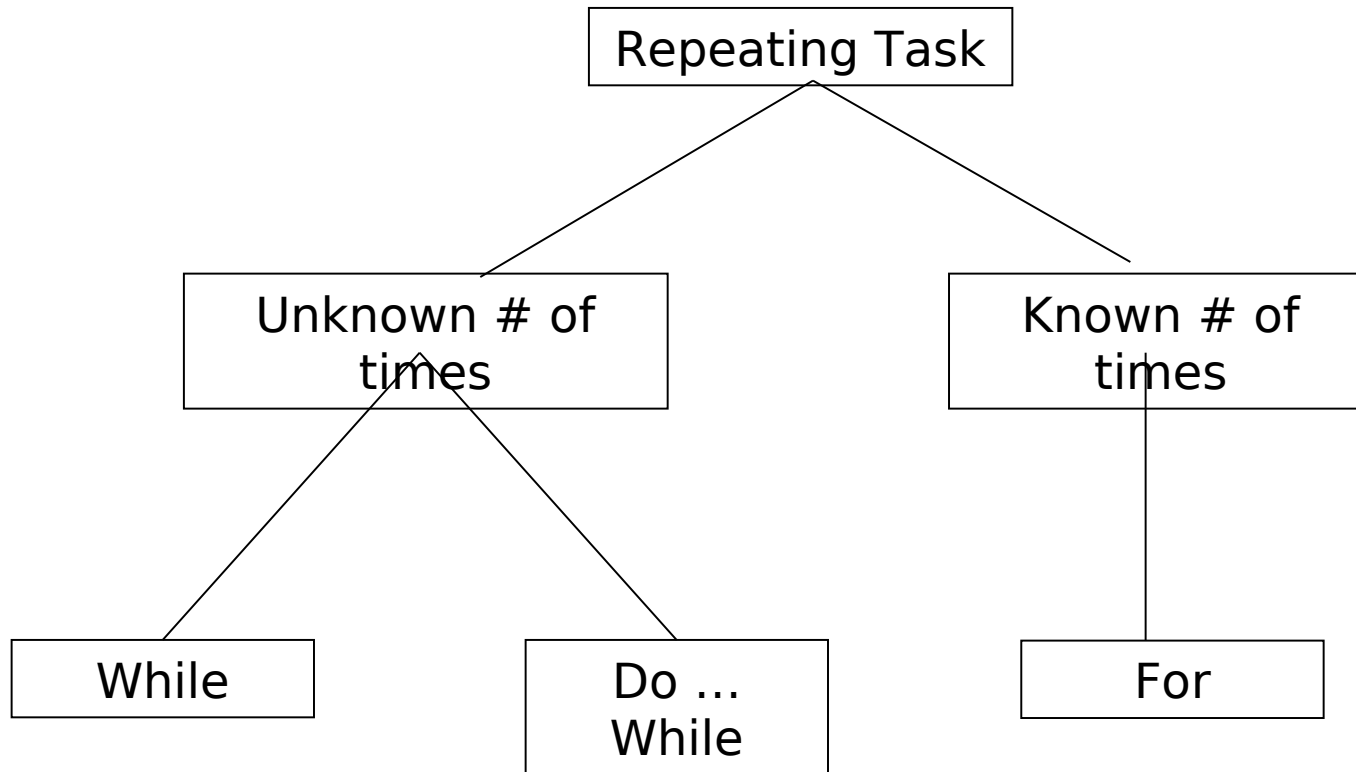
The for statement

for is more appropriate for cases where it is known in advance the number of times the repeating task is to be executed.

For Loop: Example

```
int main()
{
    int n, i, nneg, nzero, npos;
    float x;
    cout << "Enter the size of the list > ";
    cin >> n;
    nneg = nzero = npos = 0;
    for (i=1; i<=n; i++ {
        cout << "Enter an element of the list > ";
        cin >> x;
        if (x < 0) nneg++;
        else if (x == 0) nzero++;
        else npos++;
    }
    cout << "Number of -ve numbers = " << nneg << endl;
    cout << "Number of zeros = " << nzero << endl;
    cout << "Number of +ve numbers = " << npos << endl;
    return 0;
}
```

Use more appropriate loop!



Problem solving by computers (again)

- Step1: Understanding of the problem
- Step2: Algorithm Development/Design
- Step3: Coding/Program development
- Step4: Testing
- Step5: Maintenance (Not to be discussed!)

Step1: Understanding of the problem

- Rewrite the program in your own words
- Identify
 - Input
 - Output

Step1: Understanding of the problem

- Get ready to ice break into the problem statement.
- Don't be solution oriented as yet – ie. Forget the process!
- You shall be problem oriented. Much of your thinking shall concentrate on understanding of the problem and not on solution seeking.
- In other words, focus on the “What” rather than on the “How”!

Step2: Algorithm Development/Design

- Step 2.1: Solve the problem yourself by taking examples
 - “Can I solve the problem?” is the important question in this sub-step.
 - If you can't solve the problem yourself, you can't instruct the computer to solve it for you.
 - Choose examples that exposes you to different scenarios.
 - This is like a game or a challenge that you need to do it. Take the challenge and immerse into it.
 - This sub step helps you to own the problem! “owning” means feeling as if the problem is originated from you.

Step2: Algorithm Development/Design

- Step 2.2: Meta-thinking
 - Continue to experiment on the problem by taking examples. This time, however, the objective is not exactly to see whether you can solve it or not ... since you did that already in Step 2.1
 - It is rather to see yourself “How you are doing it!”.
 - Meta-thinking is like thinking about thinking. It is about consciously observing how you are thinking to solve the problem.
 - Meta-thinking helps you identify the different steps you are taking towards “solving” the problem.

Step2: Algorithm Development/Design

- Step 2.3: Propose a “seed” algorithm
 - The act of meta-thinking in Step 2.2n surely makes you mindful of certain details of the steps you are taking towards accomplishing the task (ie. solving the problem).
 - A seed algorithm is just a natural language description of the algorithm used.
 - Eg. I check whether a number is negative, zero or positive and then increment the right counter.

Step2: Algorithm Development/Design

- Step 2.4: Refine the “seed” algorithm successively
 - Now is the time to give an algorithmic shape to your seed algorithm.
 - Add variables to represent data.
 - Use pseudo-code structures (or flow-chart) to reveal as much details as possible.
 - A given algorithm might be refined a number of times.

Step3: Coding/Program development

- Transform the algorithm into a program using a programming language of “choice”.
- Make your programs readable
 - Add comments
 - Choose appropriate identifiers
 - Choose appropriate conditional and looping statements

Step 4: Testing

- Testing is not about hand tracing.
- It is running the program on computers by providing properly selected sample data.
- Analyze the output of your test run with the expected outcome of the program.

If Testing is not Ok?

If the testing shows that the program is not working properly, what do you do?

“Other” Statements

Break

Continue

Exit