Computer Programming

An-Najah N. University
Computer Engineering Department
Luai Malhis, Ph.D,

The Selection and Iteration Statements

Selection statement (if Statement)

- Gives the ability to choose which set of instructions are executed according to a condition.
- Choose among alternative courses of action

Syntax :**if (*condition*)**
***statement*;**

The if statement allows you to evaluate a *condition* and only carry out the statement if the *condition* is true (not zero).

– Example:

Read student's grade
If student's grade is greater than or equal to 60
Print "Passed"

```
int grade; cin >> grade;
if ( grade >= 60 )
    cout << "Passed";
```


`if/else` Selection

Different action is taken depends on conditions being true or false

Example:

Read studen's garde
if student's grade is greater than or equal to 60
print "Passed"
else print "Failed"

```
if ( grade >= 60 )
   cout << "Passed";
else  cout << "Failed";
```

Another example:

```
if (hours <= 40.0)  pay = rate * hours;
else pay = rate * (40.0 + (hours - 40.0) * 1.5);
```


Nested If Statements

There are no restrictions on what the statements in an if statement ca be. For example an if statement can contain another if statement.

```
if (x < 0)
   if (y != 4)
        z = y * x;
   else
        z = y / x;
else
   if (y > 4)
        z = y + x;
   else
        z = y - x;
```

In the code above first if statement contains another if else construct and the else statement contains another if else construct. Please note if no braces are used always the else statement matches the closest if.

More Examples

If Statements are Independent of each other

```
int day;       cin >> day;
if (day == 1)
      cout << "Sunday";
if (day == 2)
      cout << "Monday";
…….
if (day == 7)
      cout << "Saturday";
```

In the code above all the if statement must be evaluated. However, the cout statement is executed for only one of them depending on the value of day entered. This wastes computation time.

if … else if …. else construct

```
int day;       cin >> day
if (day == 1)
      cout << "Sunday";
else if (day == 2)
      cout << "Monday ";
……..
else if (day == 7)
      cout << "Saturday";
else
      cout << "Unknown day";
```

The code above is more efficient because when one statement evaluates to true the rest of statements are skipped.

More examples

- Compute tax based on income

Income	% Tax
< 20,000	No tax
>= 20,000 and < 35,000	20%
>= 35,000 and < 50,000	25%
<= 50,000 and < 100,000	30%
>= 100,000	35%

```
int income;
cin >> income;
if ( income < 20000 )
    printf( "No tax." );
else if (income < 35000 )
    cout << "tax = " << 0.20 * income;
else if (income < 50000 )
   cout << "tax = "<< 0.25 * income;
else if ( income < 100000 )
    cout << "tax =" << 0.30 * income;
else
    cout << "tax =" << 0.35 * income;
```


Compound Statements

– Set of statements within a pair of braces

```
if ( grade >= 60 )
   cout << "Passed.\n";
else {
   cout << "Failed.\n";
   cout << "You must take this course again.\n";
}
```

– Without braces,

```
cout << "You must take this course again.\n";
```

always executed

- Block
  – Set of statements within braces { statements }

Common Mistakes

One common mistake can occur when the == (equality) operator is confused with the = (assignment) operator.

```
int n;
cin >> n;
if (n = 3)
    cout << "n equals 3";
else
    cout << "n doesn't equal 3";
```

The if statement is always true;

Common Mistakes (2)

The null statement: if (condition);

```
int num;
cin >> num;
if (num >0);
  cout << num;
```

In this case the value of num will always be printed on the screen regardless of its value. The reason is that the "cout << num;" statements is outside the if selection.

Code Examples

- if only // read double prints negative if value is less than 0

```
double db;
cin >> db;
if (db < 0)
   cout << negative;
```

- If … else // read two numbers and print the smallest

```
int x, y;
cin >> x >> y;
if ( x < y)
    cout << x;
else
    cout << y;
```


Code Examples (2)

if … else if … else construct

```
// Code to print case of letter:
 char  ch;
 cin >> ch;
  if ( (ch >= 'a') && (ch <= 'z'))
      cout << "Small Letter";
  else if ((ch >='A') && (ch <= 'Z'))
      cout << "Capital Letter";
 else   cout << "none letter";
```

The Switch Statement

- Test expression for multiple values
- Series of **`case`** labels and optional **`default`** case

```
switch ( expression ) {
    // only works if expression evaluates to integer value

     case value1:      // taken if variable == value1
          statements
           break;       // necessary to exit switch

     case value2:
     case value3:       // taken if variable == value2 or ==value3
          statements
          break;

    default:        // taken if variable matches no other cases
       statements
       break;      // break here not necessary with last option
}
```


Examples

Read in day as a number 1,2,3,….,7 and
print it as text Sunday, Monday, Tuesday, ……., Saturday.

```
int day;
cin >> day;
switch (day) {
    case 1 : cout << " Sunday"; break;
    case 2 : cout << " monday"; break;
     case 3: cout << "Tuesday"; break;
     ……..
    case 7 : cout << " Saturday"; break;
    default : cout << "unknown day";
}
```


Examples (2)

You can have multiple statements per case

```
Int x, y;
cin >> x >> y;
   Switch (x<y){
   case 1:   cout << "x is smaller than y";
              cout << x;
             cout << y;
             cout << " The sum of x and y is " << x+y;
             break;

   default :       // case 0   makes no difference
             cout << "x is greater than or equal to  y";
             cout << x + y;
}
```


Missing Breaks

```
int n;
cin >> n;
switch (n) {
   case 1: cout << "one ";
   case 2: cout << "two ";
   case 3: cout << "three "; break;
   case 4: cout << "four ";
   default: cout << "good bye";
}// In the code above if n == 1 then one two three are printed.
```

If n == 2 two three are printed.

If n ==3 three is printed.

If n ==4 four good bye are printed.

Default location

```
char gender;
cin >> gender;
switch (gender) {
    default: cout << "Uknown gender"; break;
    case 'M':
    case 'm':  cout << "Male";  break;
    case 'F':
    case 'f': cout << "Female"; break;
}
```

The example above illustrates that

The default statement does not have to be the last statement in the cases block. It could be placed anywhere first, last or in between.

the last case may or may not have break;

Multiple case values

```
char gender;
cin >> gender;
switch (gender) {
    default: cout << "Uknown gender"; break;
    case 'M':
    case'm':  cout << "Male";  break;
    case 'F':
    case 'f': cout << "Female"; break;
}
```

The example above illustrates that if the same set of statement to be executed for more than a single case value, the case values are written following each other. Then the statement to be executed follow that last case. In the example above we print Male if the input is ether 'M' or 'm'. Print Female if the input is either 'f' or 'F'

Iterations

Definition: Iteration is a repetition structure in which a set of statements are repeated while some condition remains true

- Example

  while there are more numbers to read

  Read number and perform processing

- **`while`** loop repeated until condition becomes false

- Example

```
int product = 2;
while ( product <= 1000 )
   product = 2 * product;
```


Loop Definition

- Loops allow a group of statements to be executed over and over again.
- All loops must have:
  - loop-control variable(s)
  - body - block of statements to be executed repeatedly
  - a way for the loop to be terminated.
- Three basic loop mechanisms: while, for, and do-while.

While Loop

- Has the Syntax;

```
initial condition;
while (conditional expression)
{       statement(s)     // body of loop
}
```

- The statements comprising the body of the loop will be executed until the conditional expression evaluates to false.
- Therefore one of the statements in the body should modify the loop-control variable(s) so the loop terminates.
- While loops are pre-test loops, the condition for repetition is tested before the loop is executed.

While Loop (cont.)

- The while loop may not be executed if initial loop condition is false.
- The initial condition is optional it sets up the condition based on which the loop may or may not executed.

Loop types:

(1) Count-controlled repetition

Loop repeated until counter reaches certain value

Number of repetitions known

Example:

```
int n = 0;
while (n < 10) {
        cout << "hello";
        n++
}
```


While loop continue

(2) Sentinel value:

Loop ends when certain value reached.

Example

```
int radius;  cin >> radius; // initial condition
 while ( radius <= 0 ) {
        cout<<" Zero is not a valid radius! \n"
            <<"Please re-enter the radius.\n";
        cin>>radius;
  }
```

The loop is only executed if an invalid value is entered. An invalid value is radius <=0. Loop is ended when user enters valid value for radius.

Examples

- Read 10 int number and compute their average:

```
int x;
 int count =0;
 int  sum =0;
while ( count < 10)  {
      cin >> x;
     sum += x;
     count++;
}
  cout << "The average is " << (double) sum/ count;
```

Examples (2)

Keep reading int values until their average exceeds 1000

```
int x;                    int count =0;
double  sum =0;      double average = 0;
while (average <= 1000) {
    cin >> x;
     sum += x;
    count++;
    average = sum/ count;
}
   cout << "The average is " << average;
```


Examples (3)

Read characters until '#' is entered. Print the count of small and capital letters entered.

```
int  countsmall =0;    int countcapital =0;
char c;    cin >> c;
While (c != '#') {
    If( c >='a' && c <= 'z')
       countsmall++;
    If( c >='A' && c <= 'Z')
       countcapital++;
 cin >> c;
}
cout << "the count of small is " << countsmall << endl;
cout << "the count of capital is " << countcapital;
```


Common Errors

- Not reaching the termination condition - loop never ends. It is an infinite loop.

  int x = 1; While (x > 0) cout << "hello";
- Missing braces - only first statement is executed as body of the loop.

  int x = 0; while (x < 10) cout << "hello"; x++;
- A semicolon at the end of while line

  while (condition);

  No statement is executed - it is an empty loop and an infinite loop once it starts. Like the null statement in the If structure.

  int x =0; while (x >0); cout << "hello"; x++;

Do-While Loops

- The do-while loop is a post-test loop.
- The condition is tested after the loop is executed.
- The loop is always executed at least once.

```
do {
      statement(s)    // body of loop
} while (condition);
```

Example:

```
int x =0;
do {
      cout << x;
   } while (x !=0);
```

This loop prints 0 before it stops.

Common Use

A common use is printing menu continuously on screen:

```
int n1, n2, result;
Do {
        cout<<"Please enter S to subtract two values“ << endl
                << “enter A  to add two values” << endl
                << “or enter  Q to quit”;
        cin> > operation;
        switch (operation) {
                        case ‘A’ : cin >> n1 >>  n2;
                                        result = n1 +n2;
                                       cout << result;
                                break;
                               case ‘S’: …..
                }
    }   while (operation != ‘Q’);
```


Another Example

Keep doing area calculations for rectangles while user wishes

```
void main ( ) {
        int length, width; char answer;
        do {
                cout<< “please enter length: ”;
                 cin >> length;
                 cout << “please enter width:”;
                 cin >> width;
                 cout << “the area is “ << length * width << endl;
                cout<<“Would you like another calculation enter”
                        <<  y or Y  any other character to quit”;
                cin>> answer;
                } while (answer == ‘y’ || answer == ‘Y’);
        }
```


For-loop

- Pretest loops
- Mostly count-controlled
- Have the format

```
for(initialization; test; update)
    {     statement(s)     }
```

// suppose you wanted to read 5 numbers and print their sum

```
int j, num,  sum = 0;
for( j = 0; j < 5; j++) {
 cin >> num;
 sum += num;
}
cout<<sum;
```

- Note if j is only used to control the number of imes this loop executes, you could write for (j = 1; j <=5; j++) … or for (j = 5; j > 0; j--) ….. would give the same results.

Common Mistakes with for loop

- Missing braces – only the first statement is repeated.

```
 int sum =0;
 for (int I =1; I <=5; i++)     // note that 1+2+3+4+5 is 15
       cout << I;
       sum +=I;
cout << “          the sum is “ << sum;
```

prints: 12345 the sum is 5; why?

- Updating the loop control variable in the body of the loop – the variable is updated twice.

```
 for ( int I =0; I < 10; i++)  {
      cout < i << endl;
      i++;
}
```

Prints: 0 2 4 6 8 why?

Omitting some of the loop parts

One, two, or all of the expressions may be omitted from the for loop.

Sample 1: for(; x < 10 ; x = x+2)

the initial value of x is taken from earlier part of code.

Sample 2: for(; x < 10 ;)

the loop control variable must be updated in the body of the loop to avoid an infinite loop.

Sample 3: for(; ;) **// an infinite loop like while(1)**

Loop updates and control must be done inside loop body we will handle such loops later using break statement

What Loop should you use

- Any while loop can be converted into: do while loop or for loop and vice versa.
- The loop to use depends on the problem?
- Should the loop always execute at least once?
  - Yes ➔ do-while No ➔ while or for
- Should the loop be count controlled or sentinel controlled.
  - Count ➔ for is the most common
  - Sentinel➔ while or do-while is most common
  - For loops fit more naturally with count controlled
  - While loops more naturally with sentinel controlled
  - Do while are rarely used.

Examples

keep reading and printing chars until '#' is read.

```
// while loop
   int c ; cin >> c;
   while (c != '#') { cout << c; cin >> c;}
// for loop
   char c;    cin >> c;
   for (;c !='#';) { cout << c; cin >> c;}
// do while
 char c;   do {
                    cin >> c;  cout << c;
                  } while (c!='#');
```


Example using for

Problem: Read 10 integers and print the smallest

Solution: (algorithm)

read first number consider it as smallest
then read next number and compare it with smallest
change smallest if the next number is smaller than smallest
Repeat the process until 10 numbers are read.

```
Code :  int num, smallest;
        cin >> num; smallest = num;
        for (int i =1; I <10;  i++) {
                cin >> num;
                if (num < smallest)
                smallest = num;
}
cout << " the smallest number is << smallest;
```

Example using while

Write code that keeps reading int numbers until 0 is entered find the sum of all odd numbers and the product of all even numbers.

Algorithm:…….

```
Code: int num;  int sum = 0; int  product =1;   // initial values
    cin >> num;
    while (num !=0) {
          if ( num%2)                  // if (num%2 ==1)
                sum + = num;
          else                         // if (num% 2 ==0)
                product *= num;
          cin >> num; }
cout << "the sum of all positive numbers is  "<< sum << endl;
cout << "the product of all negative numbers is "<< product << endl;
```

Nested Loops

Some problems require the use of a loop inside another loop.

Example keep reading integers until 0 is entered and for each read integer n compute the sum of values from 1 to n inclusive

- Design and test outer loop
- Design and test inner loop

Code: for inside while

```
int num, sum;       cin >> num;
while (num > 0) {          // outer loop
    sum =0;
    for (int i =1; i < = num; i++)     // inner loop
          sum += num;
    cout << sum << endl;
    cin >> num;
}
```


More Nested Loops

For Loops can be "nested" inside another for loop

```
        for (int j = 1; j < 10; j++) // outer loop
              for (int k = 1; k<= 10; k++) // inner
                  cout << j << 'x' << k << "=" << j*k; << endl;
```

For each iteration of the outer loop the entire inner loop is executed

Another example: int i; j, sum;

```
   for ( i =1; i <=3; i++)
   {   cout << i << endl;
       cin >> j;    sum =0;
        for (j = 1; j <= n; j++)
            sum +=j;
  } loops read j and prints sum form 1 to j repeated 3 times
```


Inter loop dependence

inside loop is control dependent on outside loop

```
     int outer, inner;
    for (outer=1; outer<=5; outer++) {
          int sum =0;
         for (inner=1; inner <= outer; inner++)
                sum += inner;
          cout << sum <<  "     ";
        }
```

The output: 1 3 6 10 15

Notice not using braces because the entire inner loop is considered as one statement with respect to the outer loop.

Using "Break" in Loops

- It is a way to stop loop execution.
- It should be used very cautiously as it makes the code more difficult to understand.
- It is usually part of an if structure inside the loop

```
if (cond)  break;
```

Example: Keeps reading and printing characters until small letter is entered

```
char c;
while (1) {
    cin >> c;
    if ( c >= 'a' && c <='z'))   break;
    cout << c; }
```


Common Use

- When the programmer writes a for loop or a while loop, but wants to stop the loop when some value is reached.

```
char c; while (1) { cin >> c; if ( c =='#') break; …….}
int x; for (;;)    { cin >> x; if ( x%2) cout << x; else  break;}
```

- Example 1:

```
for (int i =0;   I < 100 ; i++) {
        cout << i;
        if (I == 9) break; }
```

Prints: 0123456789

- Example 2:

```
int x =10, sum =0;
while (--x) { sum += x; if (x ==5) break;}
cout << sum;
```

Prints: 35

Break in Nested Loops

- When break is used in an inner loop, it only interrupts that loop, the iterations of outer loops would continue.
- When break is used in an outer loop, the inner loop is also ended.

Example:

```
int j =0;
while (j < 3){
    for (int I = j; I < 10; i++) {
        cout << I;
        if (I == 5) break; }
        cout  << endl;  j++;  }
```

Prints:

```
012345
12345
2345
```


Using Continue in Loops

- It is usually used with an if structure inside the loop
- Tells the loop to skip over the remaining statements in the body and go execute the update statement
- Example 1:

```
for (int I = 0; I < 10; i++) {
    if (I %2 )
        continue;
    cout << I;
}
```

Prints: 13579

Using Continue in Loops (2)

Be carful when using continue in a while loop?

```
int I = 0;
while (I < 10) {
   if (I %2 )
      continue;
   cout << I;
      I++;
}
```

Prints 0 then goes to infinite loop

Using break and continue in same loop

Break and continue can be included in the same loop

Example 1:

```
for ( int i =0; i < 20; i++) {
if  (i ==5 || i == 11 || i == 13)
        continue;
if (I == 16)
        break;
if (I >= 4 && i < 8)
        i=9;
 cout << I << ",";
 }
```

Prints 0,1,2,3,9,10,12,14,15

Example 2 on using break continue

```
int i =0;
while (i++) {
 if (i == 2)
    continue;
 if( i == 8)
     break;
 if( i < 5)
     cout << i << ",";
}
cout << "I = " << i;
```

Prints: I = 1

```
int i =0;
while(++i) {
    if (i == 2)
        continue;
    if (i == 8)
        break;
    if ( i < 5)
        cout << i << ",";
    }
    cout << "I =" << i;
```

Prints: 1,3,4, I=8

More Example

Write code to keep reading one integer numbers and quits if number greater than 100 is entered.

For each entered number print or not prime:

```
int  num ;
while (1) {
   cin >> num;
   If(num >= 100)  break;
   for (int i = 2;  i < num; i++)
       if (num%i ==0) break;
   if( i == num) cout << num << " is prime" << endl;
   else cout << num << " is not prime" << endl;
}
```

Summation Example

Write the code to read x and y then compute the following equation :

$$\sum_{I=1}^{I=X} 2*(Y+I^2), \quad \text{where x and y > 0.}$$

Solution:

```
int x,y;
cin >> x>>y;
for (int I =1, int sum =0;  I <= x; I++)
        sum += 2 * ( Y + I * I);
cout << "The sum is " << sum;
```


Programming problems using loops

Keep reading int numbers until 0 is entered, then
for each entered number n
if n > 0 compute the sum of values between 1 and n
if n < 0 compute the product of values between -1 and n
Solution:

```
int n;    int product =1;       int sum = 0;     cin >> n;
while (n) {
   If (n > 0)
        for (int i = 0;  i < = n; i++) sum += i;
   else
        for (int i = -1;  i >=n; i--) product *= i;
  cin >> n;
}
```


More Examples

- Print all small letters.

```
for ( char c = 'a'; c <= 'z'; c++)
      cout << z << endl;
```

- Read int number n and print its factorial

```
int n, product =1;
for (int I =n; I >=1; I--)
     product * = I;
```

- Print all numbers 0 to 1000 that are even and divisible by 3

```
for (int I = 0; I <= 1000;I++)
      if(I% 2 == 0 && I%3 ==0)
            cout << I << endl;
```


Summary

- If statement can be used independent of else
- Else statement must be associated with an if
- In if … else if … else construct: when one is true no more checking is done
- Any loop can be converted to other type
- For loops are count controlled
- While loops are sentinel
- Be carful with the null statement in if and loops

  if (x > 0); cout << x; while (x > 0); x--;