

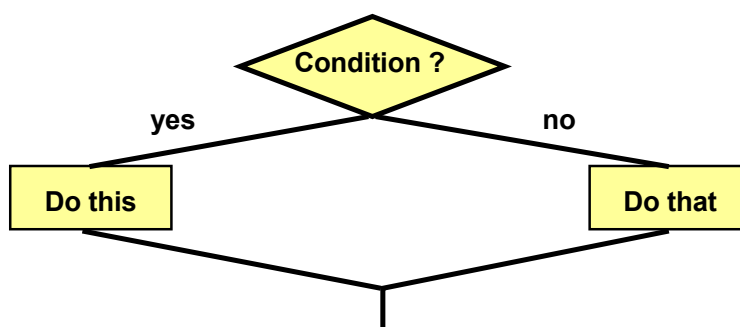
3 Selections and Loops

3.1 Introduction

In this chapter you will learn to incorporate intelligence into your programs, i.e. the program can do different things depending on different conditions (selections). You will also learn how to repeat certain tasks a specific number of times or until a specific condition is fulfilled (iteration, loop). We will introduce new symbols in our JSP graphs to illustrate selections and loops.

3.2 Selection

A selection situation can be illustrated by the following figure:



If the condition is fulfilled (yes option) the program will do one thing, else (no option) another thing.

3.3 if statement

The selection situation is in C++ coded according to the following syntax:

```
if (condition)
    statement1;
else
    statement2;
```

The keyword `if` introduces the if statement. The condition is put within parentheses. If the condition is true statement1 will be performed, otherwise statement2. Here is a code example:

```
if (a>b)
    greatest = a;
else
    greatest = b;
```

The values of two variables are compared. If a is greater than b, the variable `greatest` will get a's value. Otherwise, i.e. if b is greater than or equal to a, `greatest` will get b's value. The result from this code section is that the variable `greatest` will contain the greatest of a and b.

Sometimes you might want to perform more than one statement for an option. Then you must surround the statements with curly brackets:

```
if (condition)
{
    statements
    ...
}
else
{
    statements
    ...
}
```

If the condition is true all statements in the first code section will be executed, otherwise all statements in the second code section will be executed. Example:

```
if (a>b)
{
    greatest = a;
    cout << "a is greatest";
}
else
{
    greatest = b;
    cout << "b is greatest";
}
```

If a is greater than b, the variable greatest will get a's value and the text "a is greatest" will be printed. Otherwise the variable greatest will get b's value and the text "b is greatest" will be printed.

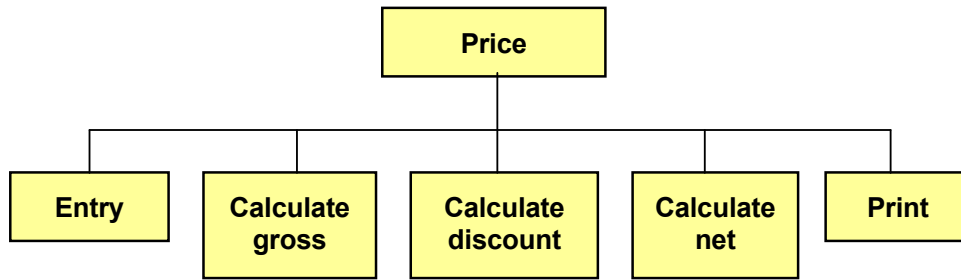
Sometimes you don't want to do anything at all in the else case. Then the else section is simply omitted like in the following example:

```
if (sum>1000)
{
    dDiscPercent = 20;
    cout << "You will get 20 % discount";
}
```

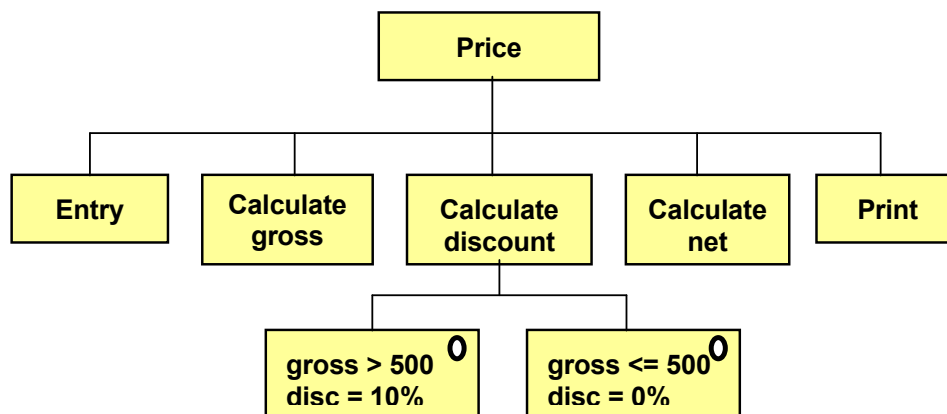
If the variable sum is greater than 1000 the variable dDiscPercent will get the value 20 and the text "You will get 20% discount" will be printed. Otherwise nothing will be executed and the program goes on with the statements after the last curly bracket.

3.4 Price Calculation Program

We will now create a program that calculates the total price of a product. The user is supposed to enter quantity and price. Download free eBooks at bookboon.com. If the total exceeds 500:- you will get 10 % discount, otherwise 0 %. We start with a JSP graph:



All boxes except “Calculate discount” are rather simple to code. “Calculate discount” requires a closer examination. It has a condition included which says that the discount is different depending on whether gross is less or greater than 500. We’ll break down that box:



“I studied English for 16 years but...
...I finally learned to speak it in just six lessons”
Jane, Chinese architect

ENGLISH OUT THERE

Click to hear me talking before and after my unique course download



A conditional situation in JSP is identified by a ring in the upper right corner of the box. That implies that only one of the boxes will be executed. Here is the code:

```
#include <iostream.h>
void main()
{
    const double dLimit = 500;
    int iNo;
    double dUnitPrice, dGross, dNet, dDisc;
    cout << "Specify quantity and unit price";
    cin >> iNo >> dUnitPrice;
    dGross = iNo * dUnitPrice;
    if (dGross > dLimit)
        dDisc = 10;
    else
        dDisc = 0;
    dNet = (100- dDisc) * dGross / 100;
    cout << "Total price: " << dNet;
}
```

The declaration shows a constant `dLimit`, which later is used to check the gross value. The variable `iNo` is used to store the entered quantity and `dUnitPrice` is used for the entered unit price.

It is common among programmers to use one or a few characters in the beginning of the variable name to signify the data type of the variable. The variable `iNo` has first character `I` (integer), and the variable `dUnitPrice` has `d` (double).

After data entry the gross is calculated by multiplying the two entered values (quantity * unit price). That value is stored in the variable `dGross`.

The `if` statement then checks the value of `dGross`. If greater than `dLimit` (i.e. 500) the variable `dDisc` will get the value 10, otherwise 0. `dDisc` contains the discount percent to be applied.

The net is then calculated by subtracting the discount percent from 100, which then is multiplied by `dGross` and divided by 100 (to compensate for the percent value).

Finally the total price is printed.

3.5 Comparison Operators

In the `if` statements in previous example codes we have so far only used the comparison operator `>` (greater than). Here is a list of all comparison operators:

```
< less than
> greater than
<= less than or equal to
```

```

>= greater than or equal to
== equal to
!= not equal to

```

3.6 Even or Odd

In some situations you will need to check whether a number is evenly dividable by another number. Then the modulus operator % is used. Below are some code examples of how to check whether a number is odd or even, i.e. evenly dividable by 2.

```

//If iNo is even, the remainder of the integer
//division by 2 equals 0:
if (iNo%2 == 0)
    cout >> "The number is even";

//If the remainder of the integer division by 2
//does not equal 0, the number is not dividable
//by 2:
if (iNo%2 != 0)
    cout >> "The number is odd";

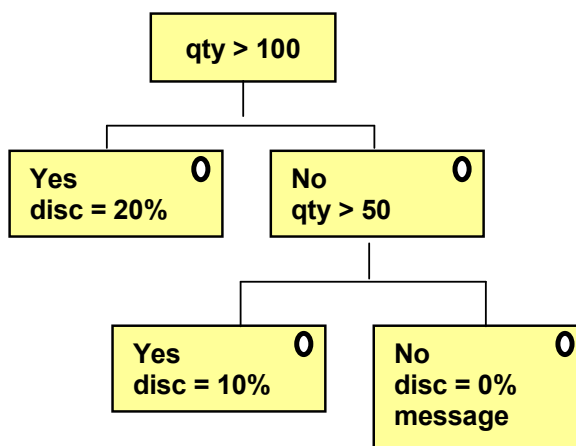
//Short way of codeing. An expression not equal
//to 0 is regarded as false, otherwise true.
//If iNo is odd, iNo%2 gives a non zero value:
if (iNo%2)
    cout >> "The number is odd";

```

3.7 else if

We will now study an example of a more complicated situation. Suppose the following conditions prevail:

If a customer buys more than 100 pieces, he will get 20% discount. Otherwise if the quantity exceeds 50, i.e. lies in the interval 50-100, he will get 10%. Otherwise, i.e. if the quantity is below 50, no discount is given. The situation is shown by the following JSP graph:



The code for this will be:

```
if (iNo>100)
    dDisc = 20;
else if (iNo>50)
    dDisc = 10;
else
{
    dDisc = 0;
    cout << "No discount";
}
```

Here we use the keyword else if.

You can use any number of else if-s to cover many conditional cases.

3.8 and (&&), or (||)

The situation with different discount percentages for different quantity intervals can be solved in another way, namely by combining two conditions. In common English it can be expressed like this:

If the quantity is less than 100 and the quantity is greater than 50, the customer will get 10% discount.

Here we combine two conditions:

- *If the quantity is less than 100*

What do you want to do?

No matter what you want out of your future career, an employer with a broad range of operations in a load of countries will always be the ticket. Working within the Volvo Group means more than 100,000 friends and colleagues in more than 185 countries all over the world. We offer graduates great career opportunities – check out the Career section at our web site www.volvogroup.com. We look forward to getting to know you!

VOLVO
 AB Volvo (publ)
www.volvogroup.com

VOLVO TRUCKS | RENAULT TRUCKS | MACK TRUCKS | VOLVO BUSES | VOLVO CONSTRUCTION EQUIPMENT | VOLVO PENTA | VOLVO AERO | VOLVO IT
 VOLVO FINANCIAL SERVICES | VOLVO 3P | VOLVO POWERTRAIN | VOLVO PARTS | VOLVO TECHNOLOGY | VOLVO LOGISTICS | BUSINESS AREA ASIA

Download free eBooks at bookboon.com



and

- *and the quantity is greater than 50*

The combination of the conditions means that the quantity lies in the interval 50-100. Both conditions must be fulfilled in order to get 10%. The conditions are combined with “and” which is a logical operator. It is written && in C++. The code will then be:

```
if (iNo<100 && iNo>50)
    dDisc = 10;
```

Suppose the situation is this:

If the quantity is greater than 100 or the total order sum is greater than 1000, the customer will get 20% discount.

Here we combine the conditions:

- *If the quantity is greater than 100*

eller

- *or the total order sum is greater than 1000*

In both cases the customer has bought so much that he will get 20% discount. One of the conditions is sufficient to get that discount. The conditions are combined with the logic operator “or”, which is written || in C++. The code for this situation will be:

```
if (iNo>100 || dSum>1000)
    dDisc = 20;
```

3.9 Conditional Input

In many situations you cannot predict what a user is going to enter. It might happen that the user enters characters when the program expects integers, or that he does not enter anything at all but just press Enter. Then you can use conditional input:

```
if (cin >> iNo)
    ...
```

To understand how this code works you must know that cin is a function that returns a value. If reading of the value to the variable iNo succeeded, the return value from cin is true, otherwise false. Here is a code section that shows how it can be used:

```
cout << "Specify quantity: ";
if (cin >> iNo)
    dTotal = iNo * dUnitPrice;
else
{
    cout << "Input error";
    cin.clear();
    cin.get();
}
```

First we prompt the user for a quantity. Then the program halts (cin) and waits for a value. If data entry turned out well, the total price is calculated.

If the data entry failed, i.e. if the user entered letters or just pressed Enter, the condition is false and the statements after else are executed. The user will get a message about input error, the keyboard buffer is cleared (`cin.clear()`) and the next character in the queue is read (`cin.get()`). This clean-up procedure must be performed to be able to enter new values to the program.

3.10 The switch statement

In addition to the if statement there is another tool that allows you to perform different tasks depending on the circumstances. The tool is called switch statement and is best accommodated to the situation when checking a value against several alternatives. A good example is a menu where the user enters a menu option (1, 2, 3 ... or A, B, C ...) to make the program do different things depending on the user's choice.

The switch statement has the following syntax:

```
switch (opt)
{
    case 'A':
        //statements
        break;
    case 'B':
        //statements
        break;
    ...
    default:
        cout << "Wrong choice";
        break;
}
```

First comes the keyword `switch`. Within parenthesis after `switch` there is the variable to be checked. It is checked against the values after the different case keywords. If for instance `opt` has the value 'A', i.e. `opt` is a char variable in the example above, then the statements below case 'A' are executed. Note that the keyword `break` must be found at the end of each case block. If `break` is omitted, the program will continue into the next case block. Note also that there must be a colon (`:`) after each case line. The default block takes care of all other options, i.e. if the variable `opt` does not contain any of the values 'A', 'B' etc. then the statements in the default block will be executed. The entire switch block should be surrounded by curly brackets.

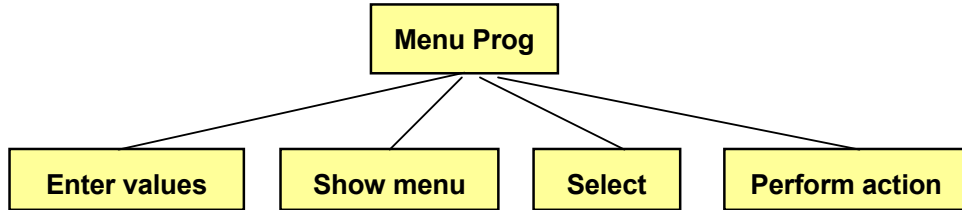
3.11 Menu Program

We will now write a menu program that illustrates how the switch statement can be used. First, the user is prompted for two numbers, and then a menu is displayed where the user can select whether to view the greatest, the least, or the average of the two numbers. The screen will look like this:

```
Enter 2 numbers: 7 5
1.  Greatest
2.  Least
3.  Average
Select:
```

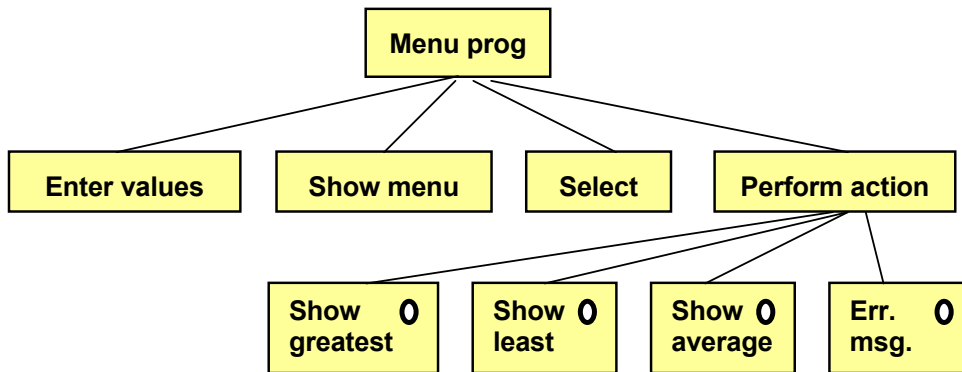

First, the user has entered the numbers 7 and 5. Then a menu is displayed where the user has to select 1, 2 or 3, depending on what he wants to view.

We will first draw a JSP graph that explains the process:



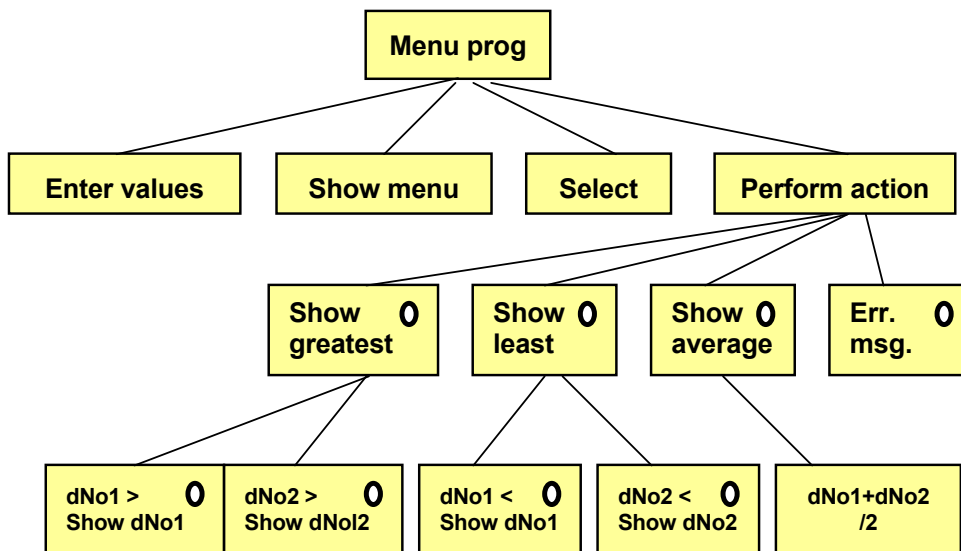
First, the user enters two numbers. Then the menu is displayed on the screen and the user selects an option. Finally the requested action is performed.

The requested action can be one of four options, so we break down the box "Perform action":



Since we have a selection situation where only one of the options should be performed, we indicate this with a circle in the upper right corner in each selection box.

The four options contain some logic, so we break down the JSP graph further:



In the “Show greatest” case we perform a check: if iNo1 is the greatest, we print it, otherwise we print iNo2. The “Show least” is analogous. In the “Show average” case we add the two numbers and divide by 2.

The code will be this:

```

#include <stdlib.h>
#include <iostream.h>
void main()
{
    int iOpt;
    double dNo1, dNo2;
    cout << "Enter 2 numbers: ";
    cin >> dNo1 >> dNo2;
    system("cls");
    cout << "1. Greatest" << endl;
    cout << "2. Least" << endl;
    cout << "3. Average" << endl;
    cout << endl << "Select: ";
    cin >> iOpt;
    switch (iOpt)
    {
        case 1:
            if (dNo1>dNo2)
    
```

```

        cout << dNo1;
    else
        cout << dNo2;
    cout << " is the greatest";
    break;
case 2:
    if (dNo1<dNo2)
        cout << dNo1;
    else
        cout << dNo2;
    cout << " is the least";
    break;
case 3:
    cout << "The average is " << (dNo1+dNo2)/2;
    break;
default:
    cout << "Wrong choice";
    break;
}
}

```

The header file `stdlib.h` is needed to be able to clean the screen with `system("cls")`, which is done after the user has entered the two values. Then we print the menu on the screen and the user enters his choice (1, 2 eller 3) to the variable `iOpt`.

The switch statement will then check the variable `iOpt`. If it is 1, the statements after “case 1” are executed. There we check which of the two numbers are the greatest and print it. In the same way the least number is printed under “case 2”. In case of 3, the average is calculated and printed. If the user has entered anything else, the default statements are executed.

3.12 Loops

We will now continue with another powerful tool within programming, that can make the program perform a series of operations a specific number of times. Sometimes, the number of times, or the number of iterations, decided from start, sometimes it depends on the circumstances. We begin with an example:

We want to print a list of the numbers 1-10 and their squares:

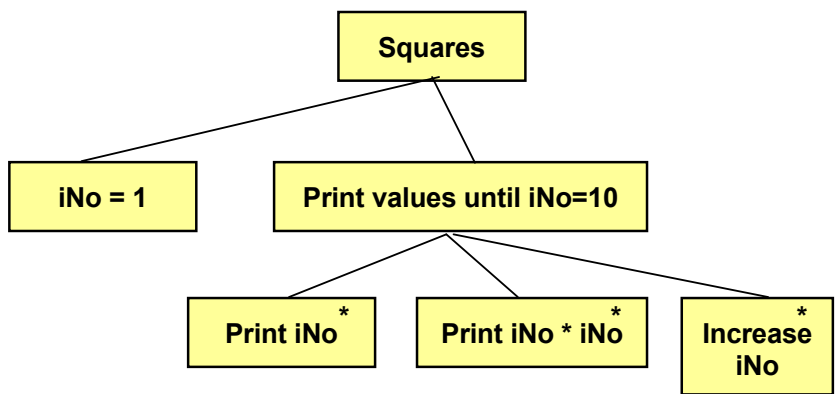
```

1 1
2 4
3 9
etc.

```

We have a variable, `iNo`, which first has the value 1. We print it and the square of it. Then we increase the value of `iNo` by 1 and repeat the process, i.e. we print `iNo` and the square of `iNo`. Then we increase `iNo` again etc. This goes on until `iNo = 10`.

Thus, we have a series of operations (print iNo, print the square of iNo) which is repeated 10 times. A repetition is called a loop. It is illustrated by the following JSP graph:



First the variable iNo is set = 1. Then a loop “Print values until iNo=10” is started. The fact that it is a loop is shown by the subordinate boxes having an asterisk in the upper right corner.

The loop consists of three boxes, which in turn print the value of iNo, the value of iNo * iNo (i.e. the square of iNo), and increase the value of iNo by 1. The loop goes on until iNo has reached the value 10.



3.13 The while Loop

Here is a code section that performs the task:

```
iNo = 1;
while (iNo <= 10)
{
    cout << iNo << " " << iNo * iNo << endl;
    iNo++;
}
```

First the variable `iNo` gets the value 1. Then the loop follows starting with the keyword `while`, followed by a condition within parenthesis. The operations to be repeated are given within the curly brackets immediately after the `while` condition.

The `while` line can be read: "As long as `iNo` is less than or equal to 10". For each turn of the loop `iNo` and `iNo*iNo` are printed on the screen, separated by a space and followed by a line break. At the end of the loop `iNo` is increased by 1.

3.14 The for Loop

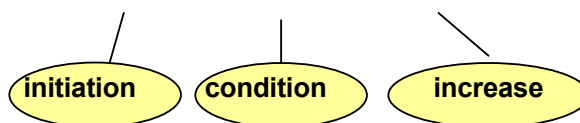
The task was solved by the `while` loop above. The `for` loop is another type of loop:

```
for (iNo=1; iNo <= 10; iNo++)
{
    cout << iNo << " " << iNo * iNo << endl;
}
```

This loop does exactly the same thing, namely prints the numbers 1-10 and their squares. The code block however contains only one statement. The actual increase of the `iNo` value is managed by the parenthesis after the keyword `for`.

The parenthesis contains three parts, separated by semicolons:

```
for (iNo=1; iNo <= 10; iNo++)
```



The initiation part sets a start value of a variable, often called loop variable, since it controls when to interrupt the loop. The condition part is checked for each turn of the loop. When the condition is false, the loop is interrupted. The increase part changes the value of some variable; mostly it is the loop variable that is increased by 1.

However, you don't have to start with 1 or increase by 1 for each turn of the loop. The following code example shows how the variable `iNo` from start is set to 2. The increase part will add 2 for each turn of the loop:

```
for (iNo=2; iNo <= 10; iNo=iNo+2)
{
    cout << iNo << " " << iNo * iNo << endl;
}
```

Download free eBooks at bookboon.com

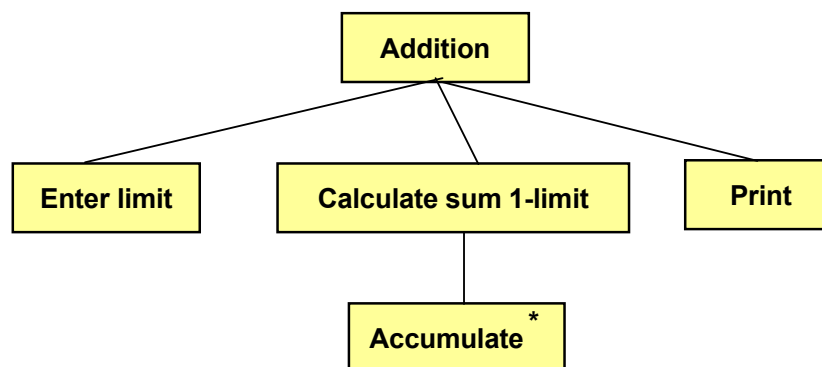
3.14.1 while or for

When should you use the while loop and when the for loop? Many times you can solve the problem with both loop types, and many times it is a question about personal preference. In general, however, if you can predict the number of turns of the loop, the for loop is the best one. If there is an unpredictable situation, e.g. if the loop goes on until the user enters a specific value, or that the random number generator provides a specific number, use the while loop. We will use both alternatives.

3.15 Addition Program

We will create a program that adds the integers $1 + 2 + 3 + 4 + \dots$ up to the limit specified by the user. The user should first enter the requested limit. Then we will use a loop that goes from 1 to that limit and sums the numbers. We will then need a variable, which is a kind of accumulator, which stores the sum.

We begin with a JSP graph:



First the user is prompted for a limit. The following loop goes from 1 to limit with the loop variable i . For each turn of the loop we add the value of i to the sum, i.e. we accumulate the numbers. Finally we print the accumulated sum. Note that the operation to be repeated (Accumulate) in the loop is indicated by an asterisk in the JSP graph.

```

#include <iostream.h>
void main()
{
    int i, iLimit, iSum = 0;
    cout << "Enter limit: ";
    cin >> iLimit;
    for (i=1; i<=iLimit; i++)
        iSum += i;
    cout << "The sum = " << iSum << endl;
}
  
```

First, a number of variables are declared. The variable i is used as loop variable, $iLimit$ is used for storage of the user specified limit, and $iSum$ the accumulated sum. Note that, since $iSum$ is increased by a value all the time, it must have a

start value. A declared variable does not automatically get the value 0 or any other value. That is why we must initiate it with 0. The other variables will get fix values during the execution of the program, so they need not be initiated.

The user will then enter a value to be stored in the variable `iLimit`. The loop then sets a start value = 1 to the loop variable `i`. The for-condition is that `i` is not allowed to exceed `iLimit`. For each turn of the loop the loop variable is increased by 1. That means that the value stated by the user controls the number of turns of the loop.

The repetition code block of the loop contains only one statement. Therefore we don't need any curly brackets surrounding the loop. If, however, the repetition code block contains several statements, they must be surrounded by curly brackets. Compare the if statement, which works in the same way.

The repetition code block contains this statement:

```
iSum += i;
```

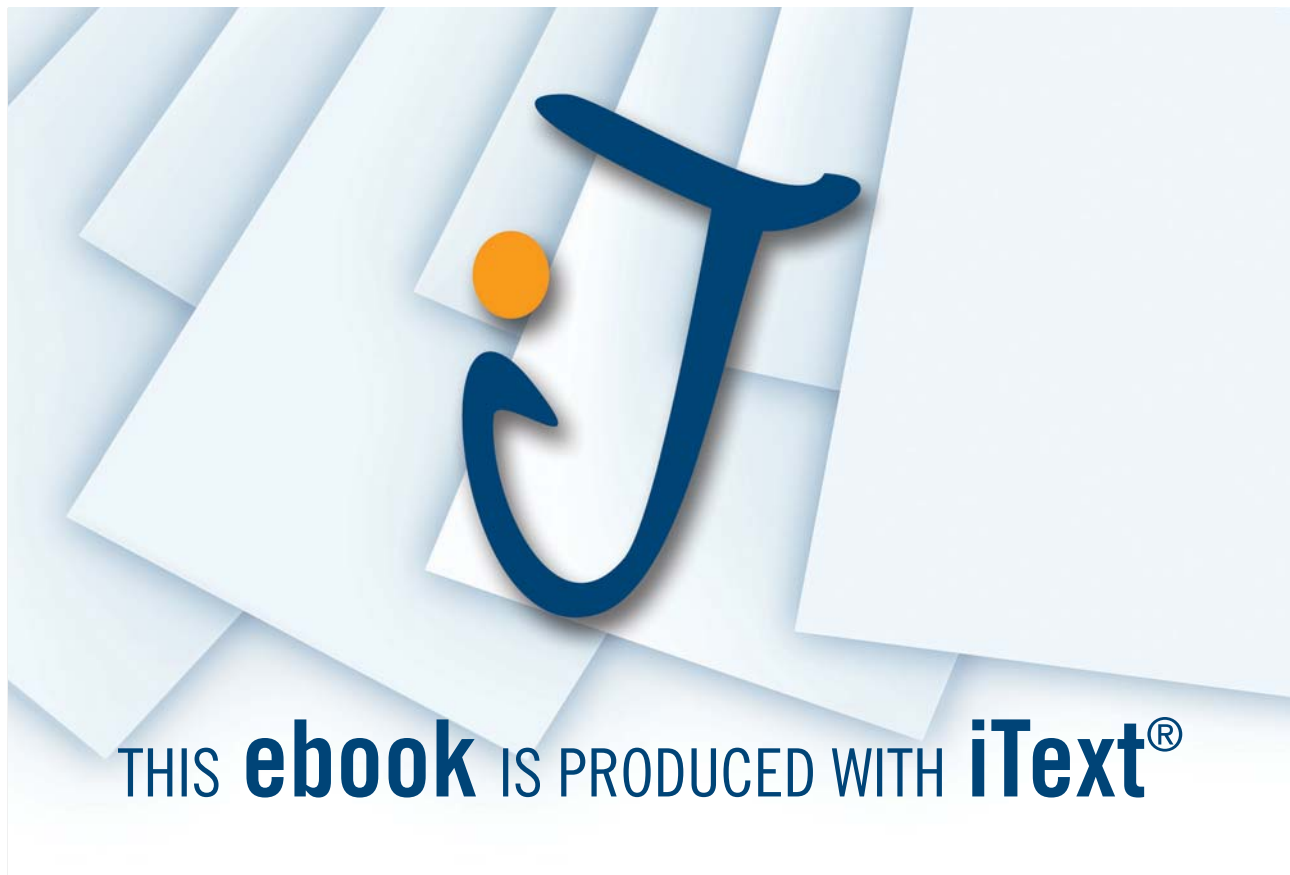
which implies that the variable `iSum` is increased by `i` for each turn of the loop. This means that the variable `iSum` will contain the sum $1 + 2 + 3 + \dots$

Finally the value of `iSum` is printed.

An optional way of writing the for line:

```
for (int i=1; i<=iLimit; i++)
```

Here we declare the variable `i` inside the for statement. The variable should then not be declared earlier in the program.



Still another way of coding:

```
for (int i=1; i<=iLimit; iSum+=i++);
```

Here the increase part of the for statement contains the code `iSum+=i++`. Here two things happen, namely that the variable `iSum` is increased by the value of `i`, and then `i` is increased by 1 (`i++`). This means that we don't need any repetition code block, so we put a semicolon directly after the parenthesis. Consequently the loop consists of one single line.

3.16 Double Loop

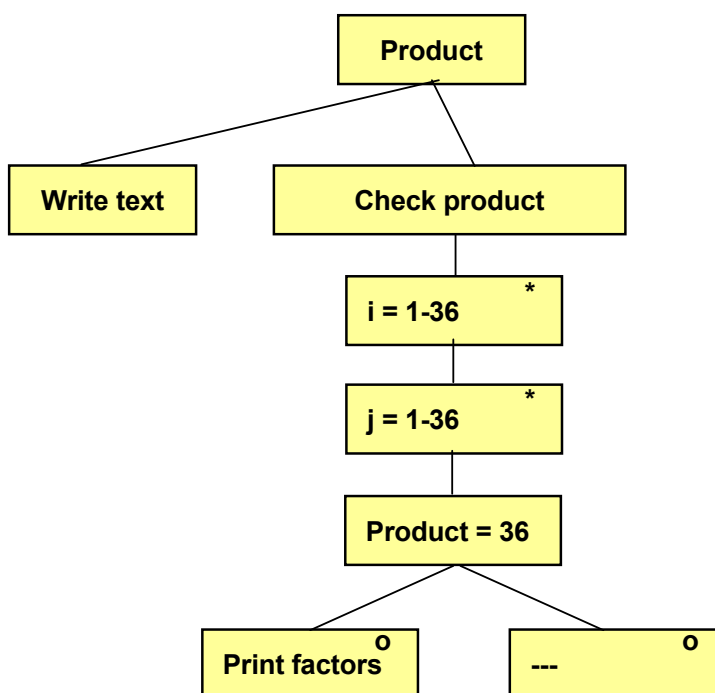
We will now how to use a double loop, i.e. a loop inside another loop. The inner loop will then do all its loop turns for each turn of the outer loop. Here is an example.

We will write a program that figures out all combinations of two integers whose product is 36:

- 1 x 36
- 2 x 18
- 3 x 12
- etc.

We let the outer loop control the first factor, which runs from 1 to 36. For each value of the first factor we will go through the values 1-36 for the second factor and check if the product equals 36. If so, the factors are printed.

First we give a JSP graph:



Under “Check product” we have an outer loop with the loop variable `i` and an inner loop with loop variable `j`. Inside the loop we check if the product of `i` and `j` makes 36. If so, `i` and `j` are printed.

Here's the code:

```
#include <iostream.h>
void main()
{
    int i, j;
    cout << "Calculation of productt" << endl;
    for (i=1; i<=36; i++)
    {
        for (j=1; j<=36; j++)
        {
            if (i*j == 36)
                cout << i << " and " << j << endl;
        }
    }
}
```

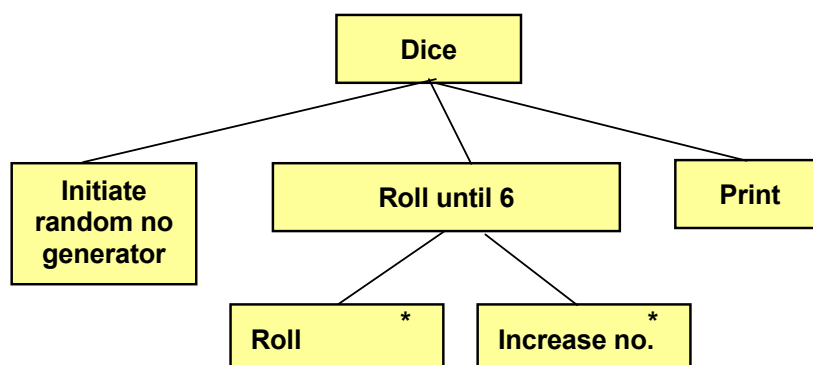
In the double for loop the variable i gets the value 1. The inner loop starts and lets j run through the values 1-36. For each value of j we check if i*j makes 36. If so, we print the values of i and j. When the inner loop has finished, the next turn of the outer loop will start where i is set =2, and the inner loop starts once again and lets j run from 1 to 36.

3.17 Roll Dice

So far we have mainly used the for loop. We will now look at a few situations where the while loop is preferred. We will write a program that rolls a dice until we get 6. Then the number of rolls is printed.

Here we cannot predict how long the loop will run. That depends on the numbers being generated. Therefore, the while loop is perfect.

Let us first create a JSP graph:

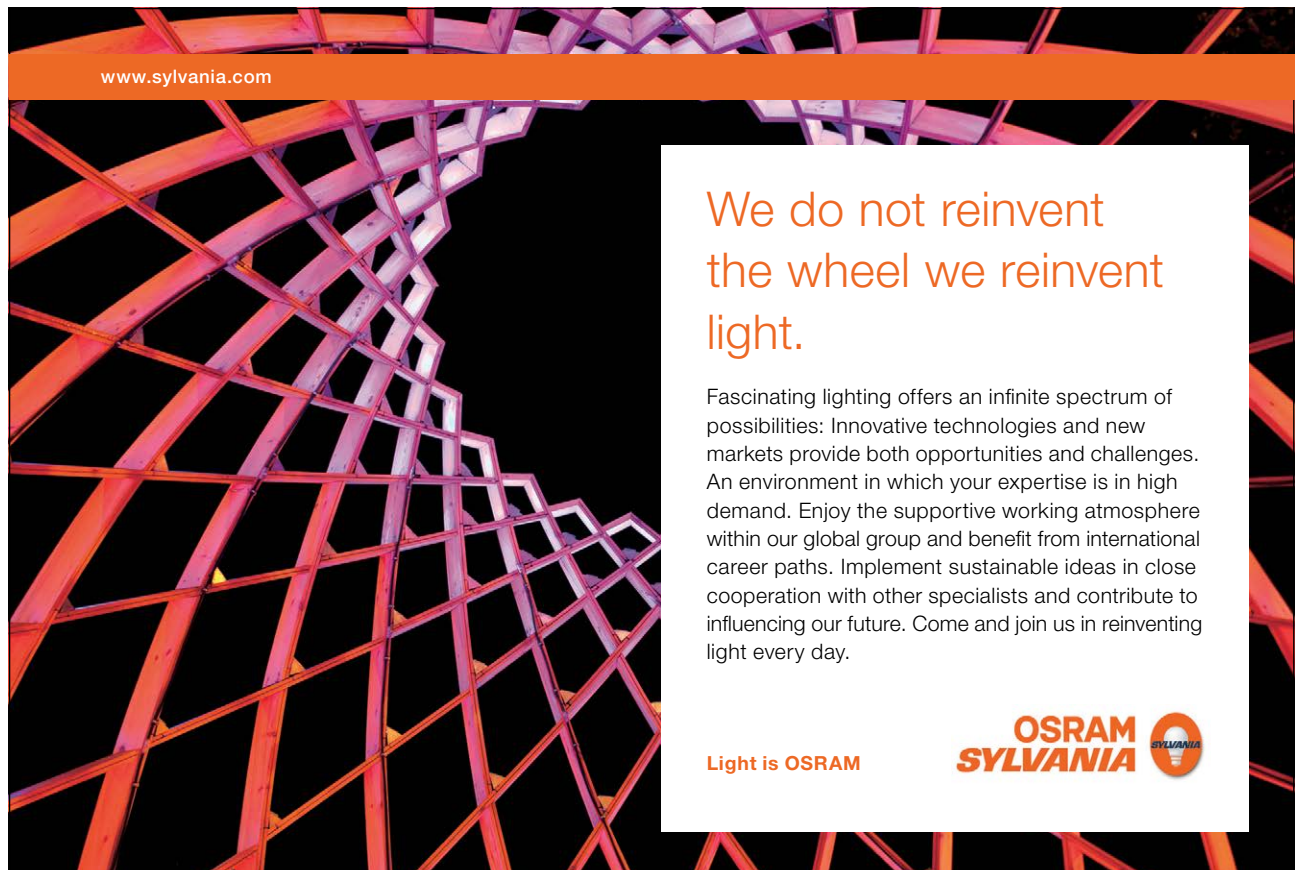


We begin with initiating the random number generator to a randomly selected start position. Then the loop is repeated until we get 6. For each turn of the loop we roll the dice once more and increase the number of rolls by 1. When 6 has been achieved, the loop is terminated and the number of rolls is printed. Here's the code:
 Download free eBooks at bookboon.com

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
void main()
{
    int iRoll=0, iNoOfRolls=0;
    srand(time(0));
    while (iRoll != 6)
    {
        iRoll = rand()%6+1;
        iNoOfRolls++;
    }
    cout << iNoOfRolls;
}
```

The header file `stdlib.h` is needed for the random number functions, and `time.h` is needed for the function `time(0)` at initiation of the generator.

The variable `iRoll` is used to store each roll. The reason for initiating it with 0 at the declaration is that it must hold a value when the while loop starts. The value must be something else than 6, otherwise the loop will not start. Any other value will do.




www.sylvania.com

We do not reinvent
the wheel we reinvent
light.

Fascinating lighting offers an infinite spectrum of possibilities: Innovative technologies and new markets provide both opportunities and challenges. An environment in which your expertise is in high demand. Enjoy the supportive working atmosphere within our global group and benefit from international career paths. Implement sustainable ideas in close cooperation with other specialists and contribute to influencing our future. Come and join us in reinventing light every day.

Light is OSRAM

OSRAM
SYLVANIA



The variable `iNoOfRolls` is used to count the number of rolls. It must be initiated to 0, since it is increased by 1 all the time.

The function `srand()` initiates the generator to a randomly selected start position.

The while loop contains the condition that `iRoll` must not be 6. As long as no 6 is achieved the loop runs one more turn. For each turn we make another roll stored in `iRoll`, and the variable `iNoOfRolls` is increased by 1.

When we get 6, the while condition is false and the loop is terminated. The variable `iNoOfRolls` then contains the number of rolls, which is printed.

A variant of the program looks like this:

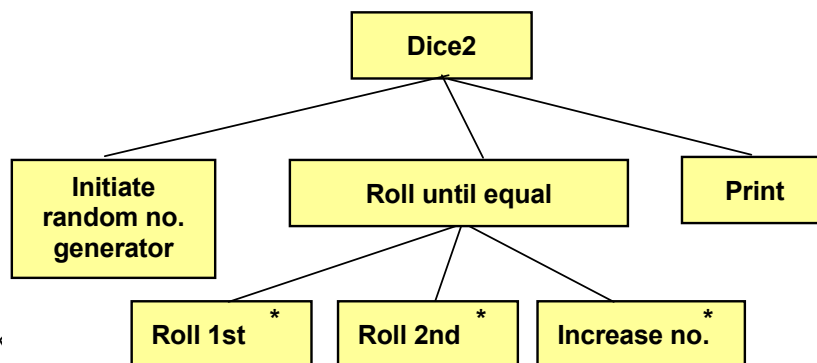
```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
void main()
{
    int iRoll, iNoOfRolls=0;
    srand(time(0));
    do
    {
        iRoll = rand()%6+1;
        iNoOfRolls++;
    } while (iRoll != 6);
    cout << iNoOfRolls;
}
```

The big difference is that the loop has its condition *after* the loop body instead of before. The effect of this is that at least one turn of the loop is executed before the condition is tested. This also means that the variable `iRoll` needs not be initialized to 0. It will anyway get a new value during the first turn of the loop.

The keyword 'do' is before the loop body, and 'while' followed by the condition right after the ending curly bracket. You must have a semicolon immediately after the condition.

3.18 Two Dice Roll

We will now write a program which repeatedly rolls two dice and checks if the two rolls are equal. When two equal rolls have been achieved, the process is terminated and the number of "double" rolls is printed. We start with a JSP graph:



When having initiated the random number generator to a random start position, the loop begins. For each turn, we roll the 1st and then the 2nd dice, and then increase the counter by 1. When the two rolls are equal the loop is terminated and the number of “double” rolls is printed. Here is the code:

```
#include <iostream.h>
#include <stdlib.h>
#include <time.h>
void main()
{
    int iRoll1, iRoll2, iCounter=0;
    srand(time(0));
    do
    {
        iRoll1 = rand()%6+1;
        iRoll2 = rand()%6+1;
        iCounter++;
    } while (iRoll1 != iRoll2);
    cout << "The rolls were " << iRoll1 << endl;
    cout << "Number of attempts = " << iCounter << endl;
}
```

The program is similar to the previous with the difference that here we have two variables which store the rolls. Inside the loop iRoll1 and iRoll2 get their values and the number of “double” rolls is increased by 1.

Since the while condition comes after the loop body, at least one loop turn will be executed. The condition is that iRoll1 is not equal to iRoll2. If they are equal the loop is terminated and the dice score and the number of rolls are printed.

3.19 Breaking Entry with Ctrl-Z

We will now use the while loop condition to contain a user input with cin. The program will prompt the user for repeated entry of numbers. The entered numbers are summed. When the user presses Ctrl-Z the entry of numbers is interrupted and their average is printed. Here is the code:

```
#include <iostream.h>
void main()
{
    int iSum=0, i=0, iNo;
    cout << "Enter a number: ";
    while (cin >> iNo)
    {
        iSum += iNo;
        i++;
        cout << "Enter one more number: ";
    }
}
```

```
    }  
    cout << "Average = " << (double)iSum/i << endl;  
}
```

The variable `iSum` is used to store the sum of the entered numbers. The variable `i` counts the number of numbers.

The while condition contains input of a number from the user. If the input succeeds, the `cin` function will return a true value. One turn of the loop is then executed. In the loop the variable `iSum` is increased by the entered value and the variable `i`, which counts the values, is increased by 1. At the end of the loop the user is prompted for yet another value.

When one turn of the loop has been run, the condition is tested again, i.e. the program halts and waits for a new entry. As long as the user enters numbers, a new loop turn is run. If the user presses Ctrl-Z the function `cin` returns a false value, which makes the loop to be terminated. Then the average is printed, which is calculated by dividing the sum by the number of values. Since the variable `iSum` is an integer we must type cast it to double before the division to not lose the decimals.

3.20 Pools

Programming a pools line (1, X or 2) with 13 football matches is another example of how to use the random number generator in a loop. Since we know that a pools line contains 13 matches, we use a for loop. First we create a JSP graph:

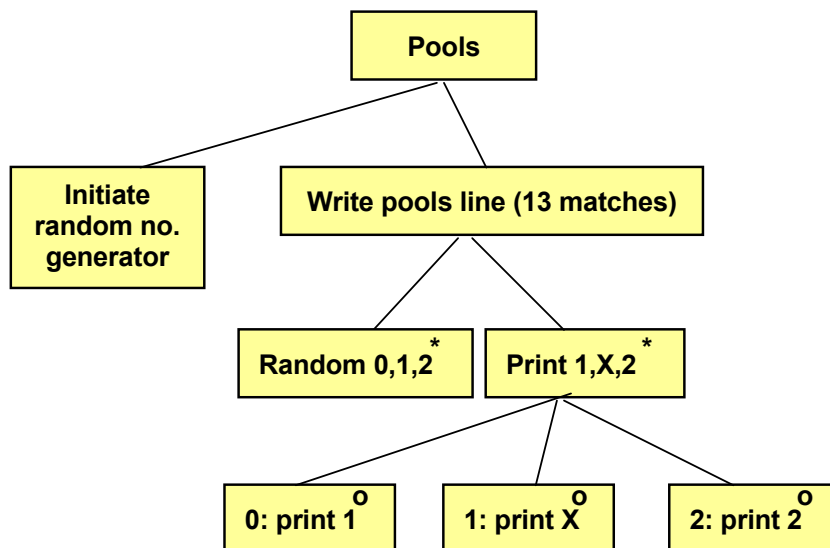


360°
thinking.

Deloitte.

Discover the truth at www.deloitte.ca/careers

© Deloitte & Touche LLP and affiliated entities.



When having initiated the random number generator, the loop begins which should create 1, X or 2 for 13 matches. We do that by create a random number which is 0, 1 or 2. If it is 0, we print '1'. If it was 1, we print 'X'. If it was 2, we print '2'. Here is the code:

```

#include <iostream.h>
#include <stdlib.h>
#include <time.h>
void main()
{
    int iNo;
    srand(time(0));
    for (int i=1; i <= 13; i++)
    {
        iNo = rand()%3;
        switch (iNo)
        {
            case 0:
                cout << "1" << endl;
                break;
            case 1:
                cout << " X" << endl;
                break;
            case 2:
                cout << " 2" << endl;
                break;
        }
    }
}
  
```

When having initiated the random number generator with `srand()`, the for loop runs from 1 to 13.

For each loop turn we create a random number in the interval 0-2, which is stored in the variable `iNo`. It is then checked by the switch statement. The different case blocks takes care of the cases 0, 1 and 2. For the value 0, we print '1'. For the value 1, we print 'X' preceded by some blanks which makes the X's appear in a separate column. For the value 2, we print '2' preceded by some more blanks.

3.21 Equation

We will now solve a math problem, namely to solve an equation of 2nd degree. To simplify, we assume that the equation has only integer roots. The equation is:

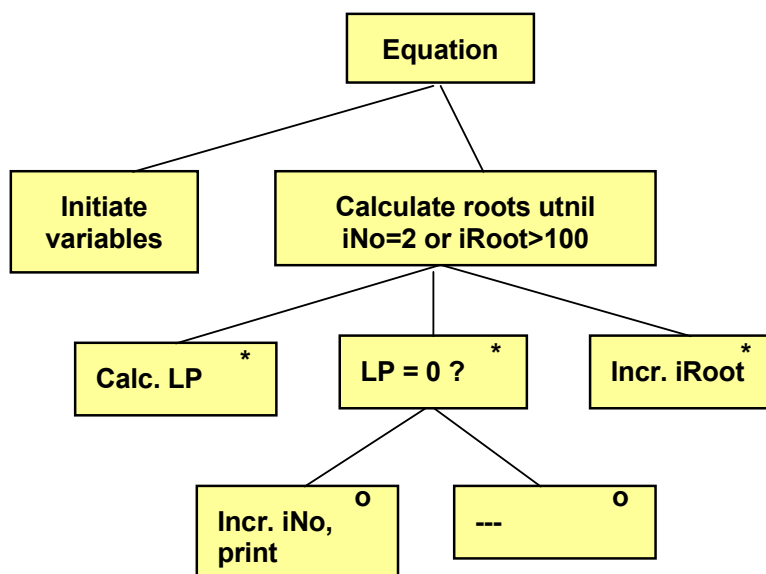
$$x^2 - 6x + 8 = 0$$

Since it is of the 2nd degree it has two roots.

Finding the solution to an equation means to find x values such that the left part (LP) equals the right part (RP), i.e. equal to 0 in our equation.

We create a loop which in turn tests the values 1, 2, 3 ... up to 100. The test procedure is to replace x by 1 in LP and calculate if LP equals 0. Then we replace x by 2 and repeat the process until we find two values that match the equation or until 100 has been reached.

First, we create a JSP graph:



The variable `iNo` is used to count the number of roots to the equation that we have got. The variable `iRoot` is the x value in the equation which in each loop turn is used to calculate LP. The value of `iRoot` starts with 1 and is increased by 1 for each loop turn. If the value of `LP = 0`, we increase the value of `iNo` and print the root (x value).

Here is the code:

```
#include <iostream.h>
void main()
{
    int iNo=0, iRoot=1, LP;
    while ((iNo<2) && (iRoot<=100))
    {
        LP = iRoot * iRoot - 6 * iRoot + 8;
        if (LP==0)
        {
            iNo++;
            cout << iRoot << endl;
        }
        iRoot++;
    }
}
```

First, the value of iNo is set to 0, since it later will be increased by 1 for each found root. The first root value to be tested is 1 (iRoot=1)..

The loop has the condition that iNo should be less than 2, since the number of roots to an equation of 2nd degree is not greater than 2, and iRoot must not exceed 100, since we don't examine roots over 100..

SIMPLY CLEVER

ŠKODA


We will turn your CV into
an opportunity of a lifetime



Do you like cars? Would you like to be a part of a successful brand?
We will appreciate and reward both your enthusiasm and talent.
Send us your CV. You will be surprised where it can take you.

Send us your CV on
www.employerforlife.com



The first statement in the loop body calculates the value of the left part (LP). This is the actual definition of the equation ($x^2 - 6x + 8$). If this value is $= 0$, we increase `iNo` by 1 and the root is printed.

Then we increase the root value by 1.

3.22 Interrupting a Loop - break

Many times you don't want to set an upper limit on the number of loop turns. Then you can use a condition for the while loop which always is true, for example:

```
while (1==1)
```

1 is obviously always equal to 1, so the loop will run an infinite number of turns. Therefore we need a possibility to, from inside the loop body, interrupt it, i.e. jump out of it and continue with the first statement after the loop. That is accomplished with the keyword:

```
break;
```

We will give a little program example of this. We will write a program where the user repeatedly is prompted for a number, and the program will respond with the square root of the number. Since you cannot calculate the square root of a negative number, we will inside the loop body check whether the user has entered a negative value. If so, the loop is interrupted. Here is the code:

```
#include <math.h>
#include <iostream.h>
void main()
{
    double dNo;
    while (1==1)
    {
        cout << "Enter a number ";
        cin >> dNo;
        if (dNo<=0)
            break;
        cout << "The square root of the number is " << sqrt(dNo)
            << endl;
    }
}
```

To be able to calculate the square root, we must include `math.h`, which contains code for a large number of math functions.

The while condition is that 1 equals 1, which always is true, i.e. we have created an infinite loop. Inside the loop body the user is first prompted for a number. If the number is less than 0, the loop is interrupted with `break`. If the number is 0 or positive, the loop goes on with calculating and printing the square root. The function `sqrt()` is used for this calculation.

Of course you could solve this problem without using an infinite loop, but regard this as an alternative to create loops.

3.23 Summary

In this chapter we have learnt to incorporate intelligence into our programs by means of selections. We have also learnt how if statements can be used to check different situations and perform different tasks depending on the circumstances. We have showed how to combine various conditions in complex situations with the operators `&&` and `||`. We have also learnt how to use the modulus operator `%` and how to perform conditional input of values from the user by placing the input statement with `cin` inside the if condition.

An alternative to the if statement is the switch statement which is often used in connection with menu programs.

We have in this chapter also introduced loops, which are used to perform a series of operations a repeated number of times. The main loops are the for loop and the while loop.

We have also extended our knowledge about the random number generator, which has been used to roll a dice and play pools game. Finally we have spent some effort by solving mathematical equations by means of loops.

3.24 Exercises

1. Write a program that prompts the user for two values and prints the least of them.
2. Write a program that prompts the user for his age. If he is younger than 15, the text "You'll got to stick to the bike some more time" should be printed. Otherwise the text "You are allowed to drive moped" should be printed.
3. Improve the previous program so that it also pays attention to the driving license age of 18.
4. Write a program that prompts the user for three numbers and prints the greatest of them.
5. Start from the Price Calculation program earlier in this chapter and apply a new discount of 5% if the gross value exceeds 250:- .
6. Continue with the previous program and write code for tax calculation, which is performed so that the user is asked for whether it is food or other products that he has bought. Let the user enter 1 for food and 2 for other products. The program should then add 12% tax for food, or 25% for other products. The tax amount and the final customer price should also be printed.
7. Suppose that the following taxing rules apply:
 - a) Income below 10 000:- is not taxable.
 - b) For income of 10 000 and more the base tax is always 50%.
 - c) For income below 50 000 a tax reduction of 5 000:- is given.
 - d) For income over 100 000 there is an extra tax addition of 20% of the portion exceeding 100 000.

Write a program that prompts the user for his income and calculates the total tax.

8. Write a program that defines whether an entered number is odd or even.
9. Improve the previous program so that it also defines whether the number could be evenly divided by 3.
10. Write a program that prompts the user for how many coins of values 0,50-crowns, 1-crown, 5-crowns and 10-crowns he has in his wallet. The program should then print the total value.
11. Write a program that prompts the user for a price. A discount percent should then be printed according to the following table:

0-100	0%
100-500	5%

500-1000	8%
1000-2000	10%
2000-5000	15%
over 5000	18%

12. Write a program that prompts the user for a quantity and a unit price of a product. If the quantity exceeds 20 and the total price exceeds 1000 kr, the user will get 20% discount. Otherwise, if either the quantity exceeds 20 or the total price exceeds 1000, he will get 10% discount. In all other cases no discount will be given. The total price and the discount should be printed.
13. Use the menu program with the switch statement earlier in this chapter and add the option:
 9. Exit
14. Extend the previous program with the option:
 4. Product
i.e. the numbers should be multiplied.
15. Write a menu program that prompts the user for three numbers and then displays the following menu:
 1. Least
 2. Greatest
 3. Sum

The program should also print the requested information.
16. Write a program that prints the numbers 1-10 and their squares.
17. Extend the previous program to also print the cubes of the values.
18. Write a program that prompts the user for integers until the entered number = 0.
19. Extend the previous program so that it also prints the sum of all entered numbers.
20. Write a program that prompts the user for integers and prints a message for each integer whether it is positive or negative. This is repeated until the entered value equals 0.
21. Write a program that prompts the user for integers until the entered value is evenly dividable by 3.
22. In many sports the competitors get scores which are the sum of the scores given by each judge after the highest and lowest score has been deducted. Write a program that prompts for one score from a judge at a time, adds the score and keeps track of the highest and lowest score. The entry is interrupted with Ctrl-Z. Then the competitor's total score should be printed after having deducted the highest and lowest score.
23. Start from the Double Loop program earlier in this chapter, which calculates pair of numbers whose product equals 36. Change it to let the user enter the product to be used.
24. Write a program that calculates the quotient of two numbers. If the quotient = 5, the numbers should be printed. All numbers up to and including 100 should be examined.
25. Start from the Roll Dice earlier in this chapter. Complete it with a printout of all rolls.
26. Change the previous program to roll the dice until it shows 5 or 6.
27. Start from the Two Dice Roll program that rolls two dice at a time. Complete it with a printout of all pair or rolls.
28. Change the previous program to roll the dice until the sum of two rolls is 12.
29. Start from the Pools program earlier in this chapter. Extend it to print 5 lines of pools beside each other, e.g.:

```

1           X           2           X           1
  X           2           1           1           X
1           1           X           2           2
etc.
```

30. Write a program that randomly prints the numbers 0 and 1. It can for instance illustrate tossing of a coin, where 0 and 1 represent the two sides of the coin.
31. Write a program that randomly pulls cards from a pack of cards and prints both colour (spades, diamonds, clubs, hearts) and value (2-10, jack, queen, king, ace). The colour and value of the card should be printed.
32. Start from the equation program earlier in this chapter. Solve the equation:
 $x^2 - 8x + 15 = 0$
33. Write a program that solves the equation of 3rd degree (3 roots):
 $x^3 - 9x^2 + 23x - 15 = 0$

I joined MITAS because
I wanted **real responsibility**

The Graduate Programme
for Engineers and Geoscientists
www.discovermitas.com



Month 16
I was a construction
supervisor in
the North Sea
advising and
helping foremen
solve problems

Real work
International opportunities
Three work placements



Download free eBooks at bookboon.com