5 Strings

5.1 Introduction

A string is a text, i.e. a sequence of characters (letters, digits and other special characters). String handling is a little tricky in C++, so we have dedicated an separate chapter to this subject.

Actually, a string is an array that consists of a number of items, where each item is a character in the string.

There are a number of string handling functions. We will in this chapter get acquainted with the most common and usable string functions, like calculating the length of a string, copying a string, concatenating strings and picking out parts of a string.

In programming, string handling is important, since a user often enters information in text form which is taken care of by the program. We will go through several programming examples, where we will have use of our string knowledge.

5.2 Data Type char

We will first get to know the most simple of all strings, namely the one containing just one character. To store one character in a variable we use the data type char. Below we declare a string variable of the char type:

```
char cLetter;
```

The variable cLetter can now contain any one character. We can assign a value to the variable:

```
cLetter = 'A';
```

Note that we use single quotes for the character.

We can also, like for all kinds of variables, assign a value directly in the declaration:

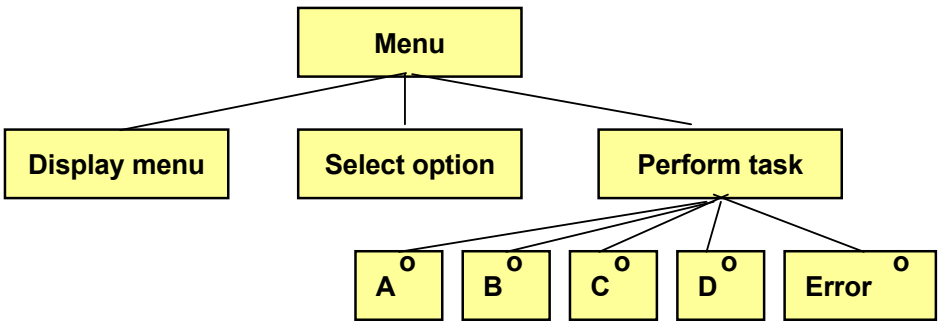
```
char cLetter = 'A';
```

5.3 Menu Program

Our first program shows how to handle entry of characters by the user to select a menu option. The program will first display a menu:

```
   Menu
   ====
A. Order
B. Invoice
C. Warehouse
D. Finance
Select:
```

Here the user can enter one of the letters A, B, C or D to choose an item. We will not build a full-featured order/invoice/ warehouse/ finance system, but we will only let the program print a text about the selected choice. We start with a JSP graph:



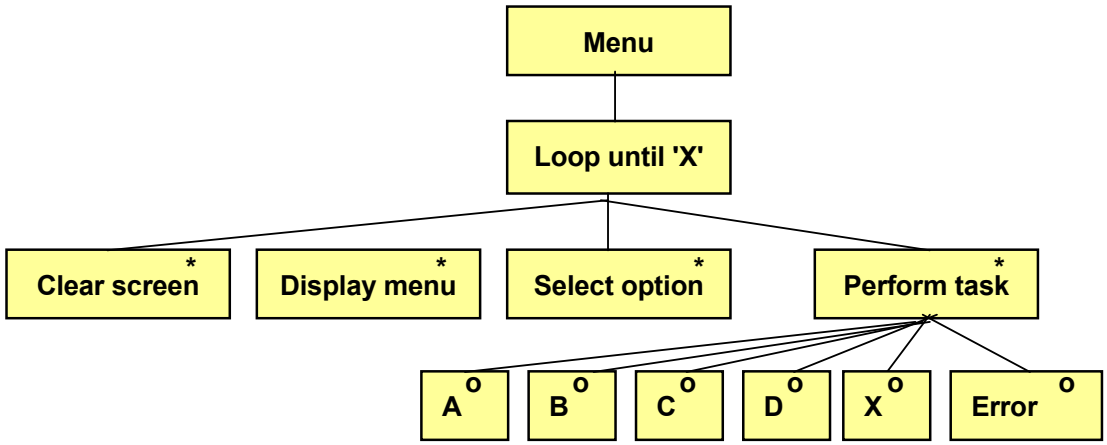
The first action is that the menu is displayed. Then the user is prompted for a choice by means of the letters A-D. Finally the requested option is executed, i.e. a simple text message will be printed. If the user enters another character than A-D, an error message is printed. Here is the code:

```
#include <iostream.h>
void main()
{
   char cSel;
       cout << "    Menu" << endl <<"    ===" << endl;
       cout << "A. Order" << endl;
       cout << "B. Invoice" << endl;
       cout << "C. Warehouse" << endl;
       cout << "D. Finance" << endl;
       cout << "Select: ";
       cin >> cSel;
       switch (cSel)
       {
     case 'A':
        cout << "You selected Order";
        break;
     case 'B':
        cout << "You selected Invoice";
        break;
     case 'C':
        cout << "You selected Warehouse";
        break;
     case 'D':
        cout << "You selected Finance";
        break;
     default:
        cout << "Erroneous choice";
        break;
   }
}
```

First a char variable is declared named cSel, and then the menu is printed with a number of cout statements. The subsequent cin statement reads a character from the user to the variable cSel, which then is checked in the switch statement. The switch statement contains one case section for each option. Note that each case line has the character A-D within single quotes, which is necessary since it is a char variable that is checked. The default section takes care of all characters other than A, B, C or D.

5.4 Menu Program with Loop

We will now improve our menu program so that the user repeatedly can enter different options without terminating the program. We then put the entire menu printing and switch statement in a loop. The JSP graph will then be:



We complete the menu with still another option, X – Exit. As long as the user does not enter X, the loop proceeds. Furthermore we also clean the screen before the menu is displayed, which is the first operation of the loop. Here is the code:

```
#include <iostream.h>
#include <stdlib.h>
void main()
{
  char cSel, temp;
  do
  {
    system("cls");
    cout << "    Menu" << endl <<"    ===" << endl;
    cout << "A. Order" << endl;
    cout << "B. Invoice" << endl;
    cout << "C. Warehouse" << endl;
    cout << "D. Finance" << endl;
    cout << "X. Exit" << endl;
    cout << "Select: ";
    cin >> cSel;
    switch (cSel)
    {
      case 'A':
        cout << "You selected Order";
        break;
      case 'B':
        cout << "You selected Invoice";
        break;
      case 'C':
        cout << "You selected Warehouse";
        break;
      case 'D':
        cout << "You selected Finance";
        break;
      case 'X':
        cout << "You selected to exit";
        break;
      default:
        cout << "Erroneous choice";
        break;
    }
    cout << endl << "Press a key to continue";
```

```
    cin >> temp;
  }while (cSel != 'X');
}
```

To be able to clear the screen we need the header file stdlib.h.

W use a do loop, where the condition is checked *after* the loop to ensure that the loop is run at least once.

The first action in the loop is to clear the screen with system("cls"). Then the menu is printed and the user is prompted for an option, i.e. a character to be stored in the variable cSel. That variable is checked in the switch statement, where a text is printed corresponding the selected option. If the user enters 'X', the text 'You selected to exit' will be printed, and the loop is terminated since the while condition specifies that cSel must not equal 'X'.

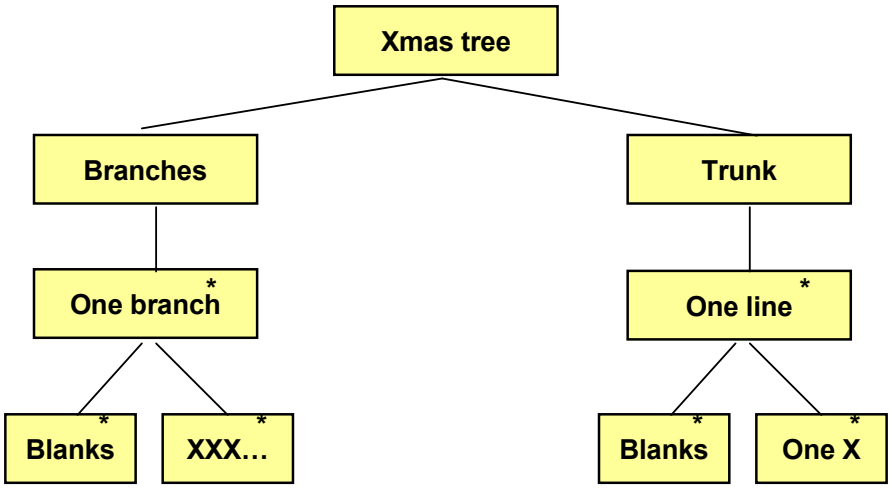
The method of letting the user enter an extra character to the variable temp at the end of the loop is a relatively unconvenient solution, but it has the advantage of avoiding to struggle with special C++ features regarding input, which we don't go into here.

5.5 Christmas Tree

We will now create a logically rather complex program that uses char variables to print a number of 'X' on the screen with the shape of a Christmas tree:

```
              X
             XXX
            XXXXX
           XXXXXXX
          XXXXXXXXX
         XXXXXXXXXXX
        XXXXXXXXXXXXX
       XXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXX
   XXXXXXXXXXXXXXXXXXX
 XXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXX
              X
              X
```

As you can see the tree has eleven branches and two 'X':s to the trunk. We therefore need an outer loop that runs eleven times, where each turn prints a branch. Each branch consists of a number of 'X':s, different depending on which branch being printed. Furthermore we will have to print a suitable number of blanks before the 'X':s, so that the branches are centered symmetrically around the middle trunk. We therefore need two inner loops, one that prints the leading blanks and one that prints the 'X':s for each branch. After completion of the branches we need a loop that runs two turns and that prints the 'X':s for the trunk. We start with a JSP graph:



The JSP graph tells us that there is one outer loop for the branches, where each turn of the loop prints a branch. We then have one inner loop that prints the correct number of blanks and one inner loop that prints the 'X':s. The same is true for the trunk where we have an outer loop where each turn of the loop prints one line of the trunk, and one inner loop that prints the blanks before one single 'X' is printed. The code is as follows:

```
#include <iostream.h>
void main()
{
   int i, j;
   char x = 'X', blank = ' ';
   for (i=10; i>=0; i--)  //first outer for-loop
   {
      for (j=0; j<i; j++)  //first inner for-loop
         cout << blank;
      cout << x;
      for (j=0; j<10-i; j++)  //second inner for-loop
         cout << x << x;
      cout << endl;
   }
   for (i=0; i<2; i++)  //second outer for-loop
   {
      for (j=0; j<10; j++)  //inner for-loop
         cout << blank;
      cout << x << endl;
   }
}
```

We declare a variable named x that correspond to the 'X', and a variable named blank corresponding to the blank character.

The first outer for-loop has an i as loop counter and goes from 10 to 0 (11 turns). We have selected to let the values run backwards to make the subsequent math easier. i is consequently a line counter for the branches of the tree.

The first inner loop has a j as loop counter and goes from 0 to i. It prints the correct number of blanks for each branch. Since i is counted reversed, the number of blanks will decrease for each branch. Each turn of the loop prints one blank.

After completion of the blanks for a bransch, first an 'X' is printed. Since the number of 'X':s at each bransch is odd, the remaining number of 'X':s to be printed is even.

The second inner for-loop prints the 'X':s with two 'X':s for each turn of the loop. We again use j as loop counter, which goes from 0 to 10-i. This means that according to the decrease of i, the number of printed 'X':s increases. And since two 'X':s are printed for each turn of the loop, there will always be an even number of 'X':s.

The second outer for-loop, which builds the trunk of the tree, has i as loop counter and goes from 0 to 1 (two turns). For each of the two turns, there is an inner for-loop that goes from 0 to 9 and that prints 10 blanks. After the inner loop, an 'X' is printed.

This complex algorithm probably requires some thinking. You could preferably write a scheme of how i and j changes their values, and in that way follow the progress of the program.

5.6 int - char

Each character has an internal code of the integer type. E.g. the character A has the code 65, B has 66 tec. Therefore the data types integer and char can cooperate. Here is an example:

```
int iNo = 65;  //The code for A
char cChr;
cChr = iNo;
cout << cChr;
```

First we declare an integer variable, iNo, and give it the value 65. On the second line we declare a char bariable named cChr. On the third line cChr gets the same value as iNo, i.e. 65. When we then print cChr on the fourth line, the program understands that it is a char variable being printed, so the character corresponding to the code 65 will be printed, namely A.

5.7 Å, Ä, Ö

As you probably has guessed, the character codes follows the alphabetic order, i.e. A=65, B=66, C=67 etc. The Swedish characters Å, Ä and Ö don't follow that pattern. Therefore you can use the hexadecimal codes for these characters:

```
'\x86'   å
'\x84'   ä
'\x94'   ö
'\x8F'   Å
'\x8E'   Ä
'\x99'   Ö
```

The characters \x indicate that a hexadecimal value will follow. If you for instance want to print the word 'från', write as follows:

```
cout << "fr\x86n";
```

If you want to assign the char variable cChr the value Ö, write this:

```
cChr = '\x99';
```

5.8 String Array, char[]

A limitation with a char variable is that it can only store one single character. Many times you want to store a longer text in a variable like a customer name or a product description. Then you will need a string array. A string array works in the same way as any other array, i.e. it has a variable name and an index value that indicates an item of the array, i.e. a specific character of the string.

Below we declare a string array called cName, which can hold up to 30 characters:

```
char cName[30];
```

The indeces can be any of 0-29.

Suppose we want the user to enter a name to the array cName:

```
cout << "Enter your name: ";
cin >> cName;
```

Let's say that the user enters 'John'. These four characters are then stored in the cName array, where the first item cName[0] contains the character 'J', the second item cName[1] the character 'o', cName[2] the character 'h' and cName[3] the character 'n'.

In addition, an extra character is always stored as the last character of a string. It is the so-called null character '\0', which is stored in the fifth position in the item cName[4].

The program should now print what the user entered:

```
cout << "Your name is " << cName;
```

In our case the text 'Your name is John' will be printed. Note that when an entire string array is printed, you don't need to specify any indeces in the cout statement. The procedure by cout is to print one character after the other in the string array from the first position until the null character is found.

Suppose that the user in the program section above enters his entire name 'John Smith'. When the cout statement is executed, still only the text 'Your name is John' is printed. The reason to this is that when text is entered from the keyboard, only characters up to the first blank will be stored in the array. This is a limitation of the cin statement.

A solution to this problem is to use another input function instead of cin:

```
char cName[30];
cout << "Enter your name: ";
cin.getline(cName,29);
cout << "Your name is " << cName;
```

Here we use the function cin.getline() instead of only cin. This has the effect that, if the user enters 'John Smith', the printout from the cout statement will be 'Your name is John Smith'. Blanks consequently work in the correct way. The input is *not* interrupted at the first blank, but the entire text line entered by the user will be stored in the variable cName.

The function getline() must have two parameters in the parenthesis; the string array to receive the entered text, and the maximum number of characters allowed to receive. The reason for using 29 is to allocate space for the null character at the end of the string as the 30th character. If the user enters less than 30 characters, say 14, the null character will be stored in position 15. If the user enters more than 29 characters, only the first 29 are accepted.

5.9 Length of a String

We will now create a little program that calculates the length of a string, i.e. the number of characters contained in the string. For that purpose we use the function strlen():

```
#include <iostream.h>
#include <string.h>
void main()
{
   char cName[30];
   cout << "Enter your name: ";
   cin.getline(cName,29);
   cout << "Your name is " << cName << endl;
   cout << "Your name is " << strlen(cName) <<
            " characters long" << endl;
   cout << "The entire string is " << sizeof cName <<
            " characters long" << endl;
}
```

To be able to use string functions like strlen() we must include the header file string.h.

In the program we first declare a string array called cName with 29 positions plus the null character, totally 30 characters. Then the user is prompted for his name, which is stored in the variable cName, and a confirmation is printed 'Your name is John Smith'.

In the next last cout statement the text 'Your name is 10 characters long' is printed, provided that we have used a name of 10 characters like 'John Smith'.

In the last cout statement the text 'The entire string is 30 characters long' is printed. Here we use the operator sizeof, which gives the declared length of the string array cName, irrespective of how many characters actually have been stored. Thus, note the difference between strlen() and sizeof.

5.10 Upper/Lower Case

We have previously mentioned that the character codes for the letters A-Z start with 65 and go upwards. This applies to the upper case characters. Lower case letters a-z have the same pattern starting with 97, i.e. 32 greater than the upper case letters. An upper case is consequently achieved by subtracting 32 from the character code of the lower case character.

We will now create a program that reads a text in lower case and converts it to upper case and prints it:

```
#include <iostream.h>
#include <string.h>
void main()
{
   int i;
   char cName[30];
   cout << "Enter your name in lower case: ";
   cin >> cName;
   int iLen = strlen(cName);
   for (i=0; i<iLen; i++)
      cout << (char)(cName[i] - 32);
   cout << endl;
}
```

As usual in string handling we must include the header file string.h.

We declare the variable i to be used as loop counter, and a string array named cName with space for 29 characters. The user is then prompted for his name in lower case, which is stored in the array cName.

We then calculate the length of the input string, which is stored in the variable iLen. This value is used as loop limit in the subsequent loop, which performs as many turns as there are characters in the input string.

Inside the for-loop we take the i^{th} character from the string cName (cName[i]) and decrease it by 32. Since this is a math operation, the program understands that it is the character code for the i^{th} character that is decreased by 32. This value is then type cast to char by placing char within parenthesis in front of the subtraction. Then the program understands that it is the corresponding character that is to be printed.

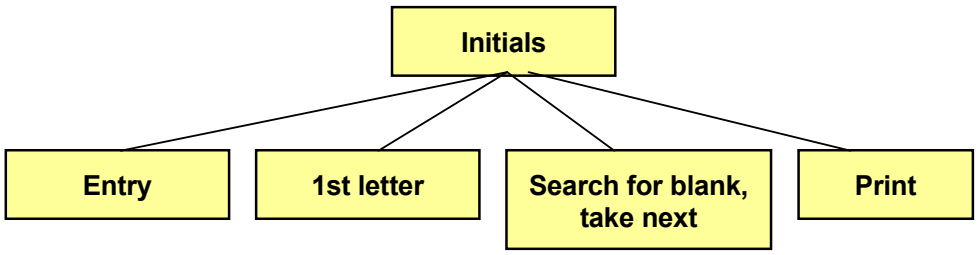
Thus, we have taken character by character from the input string, all in lower case, decreased the character code by 32, which gives the corresponding upper case character, and printed it.

5.11 Initials

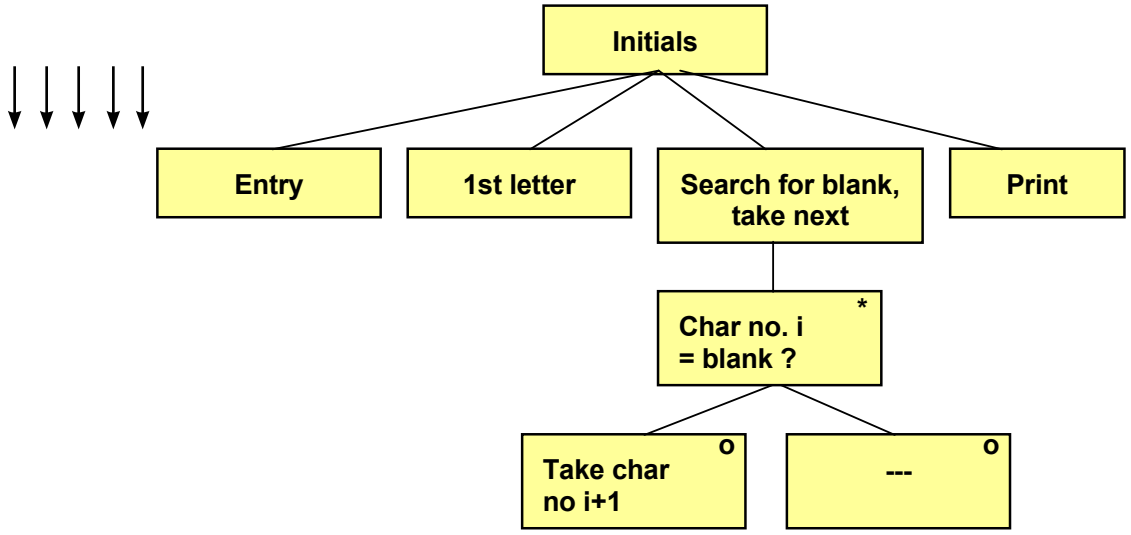
In most computer systems the programming work is to a large extent focused on checking and processing texts entered by users. Therefore, we will study some text processing examples..

In our next program example we will extract the initials from a name entered by the user. The process is to extract the first letter from the name. Then we search for the blank between the first name and the surname. In the next position the first letter of the surname is found.

We start with a JSP graph:



To search for the blank implies taking one character at a time from the beginning of the input name and check if it is a blank. We detail the 'Search for blank, take next' box:



Searching for the blank is made in a loop. For each turn of the loop we check if character no. i is blank. If so, we take character no. i+1 as the second initial. Here is the code:

```
#include <iostream.h>
#include <string.h>
void main()
{
   int i;
   char cName[30], cInit[3];
   cout << "Enter your name: ";
   cin.getline(cName,29);
   int iLen = strlen(cName);
   cInit[0] = cName[0];
   for (i=1; i<iLen; i++)
     if (cName[i] == ' ')
        cInit[1] = cName[i+1];
```

```
   cInit[2] = '\0';
   cout << "Your initials are " << cInit << endl;
}
```

We declare the string array cName, which is to contain the input name, and cInit to contain the initials. The reason for declaring 3 positions for cInit is to make space for the 2 initials plus the null character.

The user enters his name to the array cName, and its length is calculated and stored in the variable iLen.

The first character of cName (cName[0]) is stored in the first position of cInit.

Then we have the loop with the loop counter i, which represents the index in the input string, and goes from position 2 (index=1) to the end of the string. For each turn of the loop we check if the character is a blank. If so, we copy the next character (cName[i+1]) to the second position of cInit.

At completion of the loop we put the null character in the third position of cInit and print cInit.

5.12 Comparing Two Strings

In the Arrays chapter you learnt that, when comparing two arrays, you must compare each item of the arrays. For strings there is a special function, strcmp(), that can compare two string arrays. The function

```
strcmp(str1, str2)
```

- gives a negative result if str1 < str2, i.e. if str1 comes before str2 in alphabetic order

- gives 0 if str1 = str2
- gives a positive result if str1 > str2, i.e. if str1 comes after str2 in alphabetic order

We will now create a little program that prompts the user for two names and prints them in alphabetic order:

```
#include <iostream.h>
#include <string.h>
void main()
{
   char cName1[30], cName2[30];
   cout << "Enter a name: ";
   cin >> cName1;
   cout << "Enter another name: ";
   cin >> cName2;
   if (strcmp(cName1,cName2) < 0)
      cout << cName1 << endl << cName2;
   else
      cout << cName2 << endl << cName1;
   cout << endl;
}
```

First we declare two string arrays, cName1 and cName2, which can contain max 29 characters each (pos 30 is for the null character). The user enters two names. A name must not contain blanks. How would you do to allow blanks?

The if statement checks if cName1 comes before cName2 in alphabetic order. If so, cName1 is printed first and then cName2. Otherwise the names are printed in reversed order.

5.13 Copying Strings

In the Arrays chapter we copied an array's values to another array by copying each item singly. For strings there is a special function, strcpy(), that copies the entire string to another string. If for instance the string array str2 contains the string 'John Smith' and we execute the statement:

```
strcpy(str1, str2);
```

then str1 will also contain the string 'John Smith'.

5.14 Array with String Arrays

Suppose that we want to store a number of names in an array, where each name is itself a string array. Then we will need a two-dimensional array. Below we declare a two-dimensional array with space for 5 names with 30 positions each (29 characters plus the null character):

```
char cNames[5][30];
```

This two-dimensional string array could be represented like this:

	0	1	2	3	4	5	…
0	J	o	h	n	\0		
1	E	d	w	a	r	d	\0
2	B	o	b	\0			
3	E	v	E	\0			
4	A	d	a	m	\0		

To print one of the names, e.g. the third name (indes=2), we code:

```
cout << cNames[2];
```

Note that when we print or read a sequence of characters to a string array, we don't need to specify the index. The program prints the characters from the beginning of the string until the null character is found. However, for a two-dimensional array, we must specify which of the names to be printed, i.e. we must specify the first index. In the example above we used indes=2, which gives the printout 'Bob'.

To print a single character from the matrix we must use both indeces. The statement:

```
cout << cNames[1][4];
```

prints the character with index 4 from the name with index 1, i.e. 'r' in 'Edward'.

5.15 Sorting Strings

We want to sort the string matrix above, i.e. the names should be reorganized into alphabetic order. Do you remember the sorting algorithm from the Arrays chapter? If no, check it out. There we compared the items in pairs and interchanged their positions if the right item was less than the left. The logic is the same for string arrays, but we will have to use our special string functions to be able to compare and copy strings.

We first prompt the user for five names to be stored in the string matrix. Then we sort the strings in a double loop, and finally we print the sorted list of names:

```
#include <iostream.h>
#include <string.h>
void main()
{
  char cNames[5][30], temp[30];
  cout << "Enter 5 names. Press Enter after each" << endl;
  for (int i=0; i<5; i++)
    cin >> cNames[i];
  for (i=0;i<4;i++)
    for (int j=i+1; j<5; j++)
```

```
        if (strcmp(cNames[i],cNames[j]) > 0)
        {
          strcpy(temp,cNames[i]);
          strcpy(cNames[i],cNames[j]);
          strcpy(cNames[j],temp);
        }
   cout << endl << "Sorted names:" << endl;
   for (i=0; i<5; i++)
     cout << cNames[i] << endl;
}
```

First we declare a two-dimensional string array named cNames, and a string array named temp, which we will use for the triangular exchange inside the double loop.

The first for-loop reads five names to the string matrix cNames.

Then we have the double loop. The outer for-loop goes from 0 to 3 (next last name) with the loop counter i. The inner for-loop has the loop counter j which goes from 1 greater than i to 4 (last name).

The if statement compares the two names indicated by the indeces i and j. If the “left” name (i) is greater than the “right” (j), then cNames[i] comes after cNames[j] in alphabetic order, and they should interchange their positions.

The interchange is made by means of strcpy() by copying cNames[i] to the temporary string array temp. Then we copy the name in position j to position i. Finally we copy the temp name (the old name in position i) to position j. The names now have been interchanged in the string matrix.

At completion of the double loop, the names have been sorted in alphabetic order and we can with the last for-loop print one name at a time from position 0 to 4.

5.16 Substring

Sometimes you will need to extract a number of characters from a string. Then we will get use of the strncpy() function, which copies a given number of characters from one string to another. Example:

```
strncpy(cStr1, cStr2, iNo);
```

This statement copies the number of characters given by the variable iNo counted from the beginning of the string cStr2 and stores this substring in cStr1.

If you want to copy a number of characters from a given position and not from the beginning of cStr2, use this variant:

```
strncpy(cStr1, cStr2 + i, iNo);
```

This statement copies the number of characters given by the variable iNo counted from position i of cStr2 and stores this substring in cStr1.

Here we for the first time get in touch with the similarity between strings and **pointers**. A pointer is a variable that points to a certain memory address. When we write cStr2, it is interpreted as a pointer that points to the string starting with the first character of cStr2, i.e. with index=0. If we write cStr2+1, it is interpreted as a pointer that points to the character with index=1 in cStr2. If we write cStr2+i, it is interpreted as a pointer that points to the character with index=i of cStr2. Enough with pointers for this time.

We will get use of this function in subsequent program examples.

5.17 Concatenating Strings

It is possible to add strings together (concatenate) with the strcat() function. The statement

```
strcat(cStr1, cStr2);
```

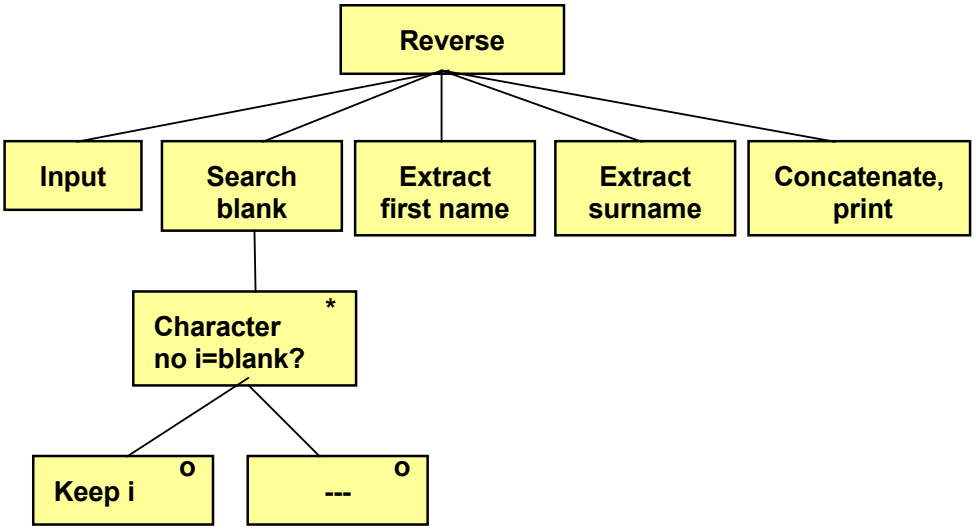
concatenates cStr1 and cStr2 and puts the result in cStr1, i.e. cStr2 is added at the end of cStr1.

We will get use of this function in subsequent program examples.

5.18 Interchanging First Name and Surname

We will now create a program that prompts the user for his first name and surname separated by a blank. The program will then interchange them so that the surname is printed first.

The way of doing this is to search for the position of the blank. With the strncpy() function we can then separate the first name and surname, and then with the strcat() function put the substrings together with the surname first. We start with a JSP graph:



First we read the name from the user input. Then we search for the position of the blank by looping through the characters and checking if any of them is blank. When a blank is found we save its position.

By means of the position of the blank we can now extract the two substrings consisting of first name and surname. Finally we put them together in reversed order and print them.

Here is the code:

```
#include <iostream.h>
#include <string.h>
void main()
{
   char cName[30], cFirst[15], cSur[30];
   cout << "Enter your name: " << endl;
   cin.getline(cName,29);
   for (int i=0; i<29; i++)
     if (cName[i] == ' ')
       break;
   strncpy(cFirst,cName,i);
   cFirst[i] = '\0';
   strcpy(cSur, cName + i + 1);
   strcat(cSur, " ");
   strcat(cSur, cFirst);
   cout << cSur << endl;
   cout << strrev(cSur) << endl; //reversed name
}
```

First we declare a string array called cName which will contain the name entered by the user (both first and surname). The string array cFirst will be used for the first name and the string array cSur to the surname. The reason for using 30 positions instead of 15 for the surname is that we will concatenate the final result in that string array.

The user is prompted for his name to be stored in the array cName.

The for-loop searchees for the blank. It goes from 0 to 28, i.e. it takes character by character from the name. If a character equals blank, we interrupt the loop with break, which has the effect that the loop counter i is not incremented any more. If the user enters 'John Smith', i will have the value 4 when the loop is interrupted, and that is the position of the blank.

The function strncpy() then copies the first i characters from cName to cFirst. In the case of John Smith the first 4 characters will be copied, i.e. 'John'.

The string cFirst is completed with a null character as the fifth character (index=4).

Then we use the function strcpy() to copy the characters from cName to cSur starting on position i+1. cName+i+1 is interpreted as a pointer that points to i+2nd character of cName. In the case of John Smith i is =4. cName+i+1 consequently points to the string starting in position with index 5, i.e. 'Smith'. Note that the copy is performed character by character until the null character is found.

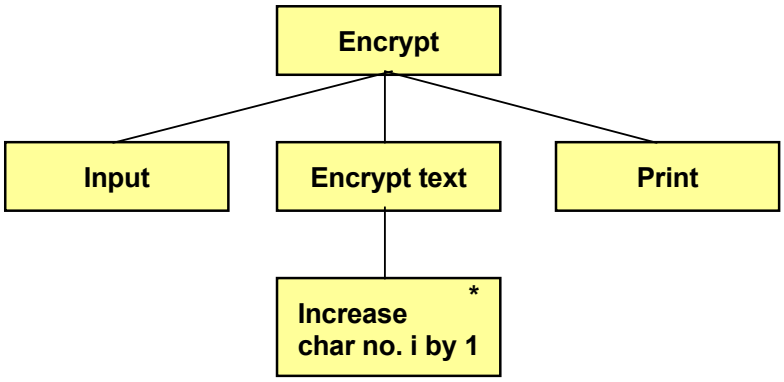
Then we use the strcat() function to concatenate the surname with the string ' ', i.e. we add a blank to the surname. Then we concatenate surname + blank with the first name. The string cSur now contains surname + blank + first name, which is printed.

The last statement has actually nothing to do with our task. We have only included it for curiosity. It prints the entire name in reversed order. The function strrev() reverses a string.

5.19 Encryption

We will now write a program that encrypts a text, i.e. distorts it to unreadability. The way of doing this is to take character by character of the text entered by the user and increase the character code by 1. A becomes B, B becomes C etc. This is a very simple encryption algorithm, but gives a hint about how different encryption algorithms work when encrypting mail and other information.

We start with a JSP graph:



The user is first prompted for a text. The encryption takes character by character from the string array and increases the character code by 1. Finally we print the encrypted message. Here is the code:

```
#include <iostream.h>
#include <string.h>
void main()
{
   int iLen;
   char cName[30], cEncrypt[30];
   cout << "Enter your name: " << endl;
   cin.getline(cName,29);
   iLen = strlen(cName);
   for (int i=0; i<iLen; i++)
     cEncrypt[i] = cName[i] + 1;
   cEncrypt[iLen] = '\0';
   cout << "Encrypted: " << cEncrypt << endl;
}
```

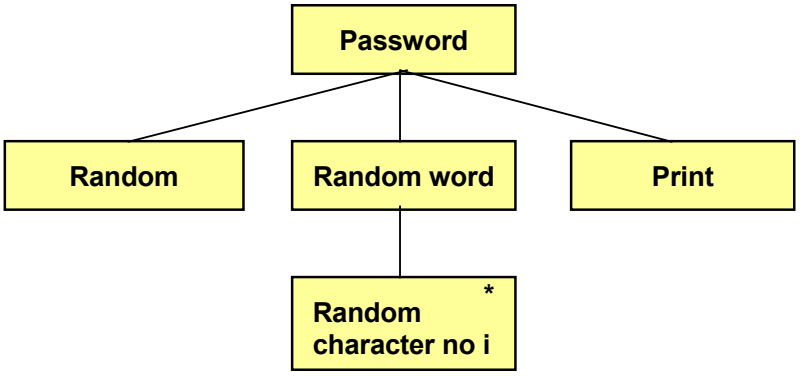
We declare the string array cName to be used for the entered name, and cEncrypt to be used for the encrypted text. The user is prompted for his name, which is stored in the string array cName. The length of the text is calculated and stored in the variable iLen.

The for-loop goes from the first to the last character of the entered string and increases the character code for each character with 1. The character is stored in the string array cEncrypt.

As the last character we insert the null character at the correct position in cEncrypt, which then is printed.

5.20 Random Password

Passwords should, as you probably know, be difficult to guess to prevent unauthorized access to a computer system. A good way of creating passwords is to let the computer randomly create them. We will write a program that makes the computer able to create passwords only consisting of upper case letters (A-Z). This is of course a limitation, but indicates the logic to be followed. You can then improve it to include lower case letters and digits. We start with a JSP graph:



We begin with creating a random length of the password, which we in this example will limit to between 5-7 characters. Then we create as many characters in the interval A-Z (character codes 65-90). The password is then printed. Here is the code:

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
void main()
{
   int iLen, i;
   char cPw[30];
   srand(time(0));
   iLen = rand() % 3 + 5;
   for (i=0; i<iLen; i++)
      cPw[i] = rand() % 26 + 65;
   cPw[iLen] = '\0';
   cout << "Password: " << cPw << endl;
}
```

We declare the loop variable i, the variable iLen to contain the length of the password, and the string array cPw to contain the random password. We have declared 30 characters, but 9 would suffice in this example.

The random number generator is initiated with the function srand(). Then we create the random length of the password with the modulus operator %, which gives a length in the interval 5-7. rand()%3 gives a random number in the interval 0-2. By adding 5 we transform the interval to 5-7. The number is stored in the variable iLen.

The for-loop then goes from 0 to iLen-1, i.e. as many turns as the number of letters in the password. For each turn of the loop a random number in the interval 65 to 65+25 is created, which corresponds to the character codes for A-Z. Each character code is stored in the string array cPw at position i, i.e. cPw is built up with one more character for each turn of the loop. At the end of cPw we add the null character. cPw is then printed.

5.21 Translation Table

We will now show another method of encrypting a message, namely by means of a translation table. It consists of a set of the characters A-Z, very well mixed up. Thus, the characters are not in alphabetic order. We use the random number generator to mix the characters.

Suppose we have got the following translation table:

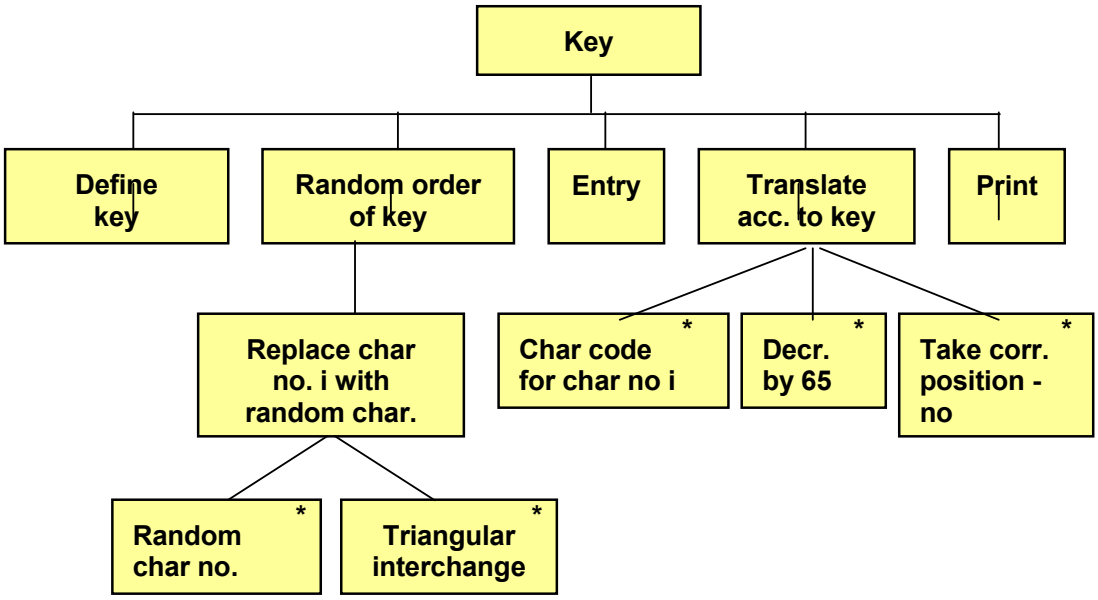
```
DGZCW...
```

where D is in position 0, G in position 1 etc. We use the table to translate a A to D, B to G etc. each character in the original message will consequently be translated to another character:

```
ABCDE...
```

DGZCW...

The JSP graph below shows the logic:



'Define key' means that we define a string containing the characters A-Z.

'Random order of key' implies that we mix the characters of the key, which is done by looping through the characters and create a random position for another character of the key. These two characters will then interchange their positions. In that way all characters will interchange its position with some other character.

The user is then prompted for the message in upper case.

'Translate acc. to key' is made by taking the character code for one character at a time in the entered message and decreasing the code by 65. A will then get the value 0, B the value 1 etc. This value gives the index in the key for the character to substitute the original character.

If we use the key:

```
DGZCW...
```

like in the introductory example, D has index 0 in the key, G index 1 etc. If the original message contains a B (character code 66), it is decreased by 65 to the value 1. This value is used as index in the key, which gives the character G.

Here is the code:

```
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
void main()
{
```

```
  int i, j, iLen;
  char cTemp, cText[50], cEncrypt[50];
  char cKey[27] = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
  srand(time(0));
  for (i=0; i<26; i++)
  {
    j = rand() % 26;
    cTemp = cKey[i];
    cKey[i] = cKey[j];
    cKey[j] = cTemp;
  }
  cout << "Write a text in upper case: " << endl;
  cin.getline(cText,49);
  iLen = strlen(cText);
  for (i=0; i<iLen; i++)
    cEncrypt[i] = cKey[cText[i] - 65];
  cEncrypt[iLen] = '\0';
  cout << "Encrypted: " << cEncrypt << endl;
}
```

First we declare some supporting variables; i, j and iLen, the latter being used for the length of the entered message. The string array cText will store the entered text, and cEncrypt for the encrypted message. The string array cKey is initated with the characters A-Z. Then the random number generator is initiated.

The first for-loop performs the mixing of the characters of the array cKey. The loop counter i goes from 0 to 25 and points to character by character in cKey. A j value is randomly created to be used as index to the character to be interchanged with the i-character. Then the triangular exchange of these two characters is performed. At completion of the loop all characters in cKey has been interchanged with some other character, which makes cKey now be well-mixed.

The user is prompted for a message that is stored in the string array cText. The length of the message is calculated and stored in the variable iLen.

The second for-loop performs the translation to an encrypted message. The loop counter i goes from 0 to iLen-1, i.e. as many turns as there are characters in the input string. In the statement

```
cEncrypt[i] = cKey[cText[i] - 65];
```

a character is taken from cText (cText[i]).. This value is decreased by 65, and since it is a math operation, the result is of integer type. If the character is B (character code 66), the calculation 66-65 is performed which gives 1. This value is now used as index in cKey and gives the character from cKey that corresponds to the original position of B, for instance G. This character is stored in the string array cEncrypt at position i. The array cEncrypt is thus built up by one character at a time from the array cText, translated via cKey. At completion of the loop all characters in cText have been translated via cKey and been stored in cEncrypt.

After the last character in cEncrypt we add the null character. The array cEncrypt containing the encrypted message is then printed.

5.22 Summary

In this chapter we have introduced the char data type. A char variable can contain only one character. We have also learnt how to use string arrays or string variables to store texts consisting of more than one character.

We have also shown how to use character codes to manipulate texts, for instance to convert between upper and lower case.

Blanks is a problem at entry of texts, which is solved by using the function getline().

The header file string.h contains a number of nice functions for text management, for instance to calculated the length of a text, copy texts, concatenate texts and extract substrings of texts. We have also created a primary relation to pointers, which we will return to more in detail in a later chapter.

We have also worked through a number of programming examples to extract initials from a name, sort a list of names, where we got use of two-dimensional arrays (matrices), interchange first name and surname, encrypt messages and create random passwords.

5.23 Exercises

1. Start from the menu program earlier in this chapter and add an extra option to the menu:
   `E. Statistics`
   Complete the switch statement with a suitable text.

2. Write a menu program where you by means of entry of a character can select to get either your name, address, postal address or telephone number printed. It should be performed with a loop so that you repeatedly can select from the menu until you enter X to exit the program.
3. Write a program that asks for a character and then prints the character 10 times at the same line.
4. Change the previous program so that the program prints the character once at the first line, twice at the second line etc. to 10 times at the tenth line.
5. Write a program that prompts the user for a character and an integer. The program should then print the character as many times as given by the entered integer.
6. Change the Christmas Tree program in this chapter so that it prints 15 branches instead of 11.
7. Write a program that prompts the user for a character and prints the corresponding character code.
8. Complete the previous program with entry of a character code and a printout of the corresponding character.
9. Write a program that prints the text 'Här går vi över ån efter vatten' with the Swedish characters å, ä and ö in a correct way.
10. Write a program that reads a text from the keyboard and prints the length of the text as well as the entire declared size of the string variable.
11. Start with the Upper/Lower Case program and change it so that it reads a text in upper case and prints it in lower case. What happens if you enter other characters. Explain why the printout is the way it is.
12. Write a program that prompts the user for a word and prints each character doubled (twice).
13. Write a program that prompts the user for a word and prints it reversed, first by using the strrev() function, and then without strrev().
14. Start from the Initials program in this chapter and complete it so that the user can enter first name, middle name and surname and get all three initials printed.
15. Complete the previous program so that, if you enter the names with leading lower case characters, the printout will still be with the initials in upper case.
16. Write a program that prompts the user for two characters and prints all characters in between. For instance, if you enter F and K, the printout should be FGHIJK.
17. Write a program that prompts the user for two words and prints the one first in alphabetic order.
18. Write a program that prompts the user for a word in lower case. The program should respond with the number of vowels in the word, Don't include å, ä or ö.
19. The Robbery language is spoken by doubling each consonant and putting an 'o' in between. For instance the word 'bug' becomes 'bobugog'. Write a program that reads a word and prints it in the Robbery language.
20. Write a program where the user can enter 6 product names. Sort the list and print it.
21. Complete the previous program so that the user also can enter the prices of the 6 products. The printout should contain the correct price for each product, right-aligned with two decimals.
22. Write a program where the user can enter his first name and surname. The program should then print only the surname.
23. Write a program where the user can enter his e-mail address. The program should then check whether there is an @ in the address.
24. Complete the previous program so that, if there is no @, the text '@htu.se' is added and the address is printed.
25. Suppose that a person enters his e-mail address with a period between the first name and the surname, e.g.: john.smith@htu.se

Write a program that reads an e-mail address and prints the name in the correct way, e.g.:
John Smith
with upper case first character in both first and surname and with a blank in between.

26. Start with the Encryption program in this chapter. Change so that B becomes A, C becomes B etc.
27. Change the previous program so that a character is replaced by the one 3 positions earlier in the alphabet.
28. Write a program that decrypts instead of encrypts according to the method in the previous exercise.
29. Write a program that encrypts a text by reversing the alphabet, i.e. A becomes Z, B becomes Y etc.
30. Start with the Random Password program in this chapter. Complete it with also taking lower case characters and digits into account.
31. Change the previous program so that the password will be 6-10 characters in size.
32. Originate from the Translation Table program in this chapter. Complete it with also taking lower case characters into account.
33. Use your imagination to construct an own encryption algorithm and write a program for it.