9 Structures

9.1 Introduction

When working with data from files and databases it is often convenient to process big portions of data in one lump, for instance an entire customer record in a customer file. A good tool for this is the structure concept. A structure is a set of data that in some way has an intermediary relation.

In connection with structures we will be using pointers and pointer arithmetics that we learnt in the previous chapter.

Structures are a pre-state to classes within object oriented programming. Therefore, this chapter is a bridge to the next step of your programmer education, object oriented programming.

In this chapter we will learn how to define structures, handle information stored in structures, work with arrays of structures and files in connection with structures. We will also learn how to use pointers to structures, how to sent structures to a function and store structures in the dynamic memory.

9.2 What Is a Structure

Think of a customer record in a customer file that contains name, address, telephone, email, discount profile, terms of delivery, terms of payment and so forth. All this information is stored for each customer in the customer file.

When reading, processing and saving this information to a file or database it is convenient to be able to handle all data for a customer in a uniform way. It is then gathered into a structure, which provides better organization of the program code.

A structure is like a template for all information per customer. A structure behaves in the code like a data type such as int, double or char. You declare a variable of the structure type defined. In the structure variable you can then store all information for a particular customer.

You can also create an array of structure items, where each item of the array is a structure with all information per customer. The array will thus contain all information for all customers.

9.3 Defining a Structure

First we will learn to define a structure template, i.e. specify the shape of the structure, the structure members and the data type of each member of the structure. Suppose we want to work with a product file with:

- Product name
- Product id
- Price
- Quantity in stock
- Supplier

This means that each product in the file will contain these five members.

Here is the code for definition of the structure:

```
struct Prod
{
  char cName[20];
  int iId;
  double dPrice;
  int iNo;
  char cSupp[25];
};
```

First there is the keyword struct, and then the name of the structure or data type (Prod). Within curly brackets you then enumerate the members of the structure, where each member is declared in the usual way of declaring variables. Each member is ended with a semicolon. After the last right curly bracket there must also be a semicolon.

The structure above shows that the different members can be of different data types (char, int, double) and also arrays like cName. You can also have other structures as members of the structure, if applicable.

The names of the structure and members are of course arbitrarily selected, but they should in some way correspond to their usage.

9.4 Declaring and Initiating Structure Variables

To declare a structure variable, i.e. a variable of the data type Prod, you write:

```
Prod prodOne;
```

Here we declare a variable prodOne which is of the Prod type. You can also initiate it with values already in the declaration:

```
Prod prodOne = {"Olive Oil", 1001, 120.50, 250, "Frescati Oil S/A"};
```

Within curly brackets we enumerate values for the structure members in the correct sequence, separated by commas. The data types of the values must correspond to the definition of the members.

9.5 Assigning Values to Structure Members

When updating, copying or in other ways processing the value of a structure member, you use the following way of coding:

```
prodOne.iNo = 251;
```

You write the name of the structure variable, followed by a period and the name of the member in question. Here the quantity in stock will be set to 251 for the 'Oliv Oil' product. Or:

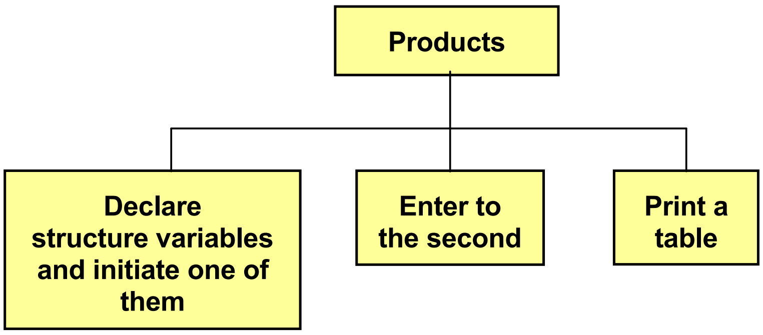
```
strcpy(prodOne.cSupp, cString);
```

This requires that cString is a string array whose content is copied to the cSupp member.

9.6 A Structure Program

We will now create an entire program using structures. We will create a product structure according to the previous example and two structure variables with product information. One of them should be initiated directly in the declaration and the other is supposed to be supplied with information from the user. Finally the program should print a table of the products.

We start with a JSP graph:



The logic is simple. The most difficult task is to handle the structure in the correct way. Here is the code:

```
#include <iostream.h>
struct Prod
{
   char cName[20];
```

```
  int iId;
  double dPrice;
  int iNo;
  char cSupp[25];
};
void main()
{
  // Declare and initiate a variable of type Prod
  Prod prodOne = {"Olive Oil", 1001, 120.50, 250,
     "Frescati Oil S/A"};
  // Declare a new Prod variable
  Prod prodTwo;
  // Prompt the user for product information
  cout << "Enter information for a product:" << endl;
  cout << "Start with the product name: ";
  cin.getline(prodTwo.cName, 20);
  cout << "The product id: ";
  cin >> prodTwo.iId;
  cout << "The price: ";
  cin >> prodTwo.dPrice;
  cout << "How many items are there in stock? ";
  cin >> prodTwo.iNo;
  cin.get();  // clear in-buffer from new line char
  cout << "Who supplies the product: ";
  cin.getline(prodTwo.cSupp, 25);
  cout <<"Prodname  \tProduct id \tPrice \tQuantity
    \tSupplier" << endl;  // tab with \t
     cout << prodOne.cName << '\t' << prodOne.iId
    << "\t\t" <<  prodOne.dPrice << '\t' <<
    prodOne.iNo << '\t' <<  prodOne.cSupp <<
    endl << endl;
     cout << prodTwo.cName << '\t' << prodTwo.iId <<
    "\t\t" << prodTwo.dPrice << '\t' << prodTwo.iNo <<
    '\t' << prodTwo.cSupp << endl << endl;
}
```

The definition of the structure is before main(), which makes it valid for the entire program, also inside functions. You can also define the structure inside main(), but then it is only valid in main() and not in other functions.

The first structure variable prodOne is initiated with values directly in the declaration. Then there are a number of heading texts and entry of values to the structure members of the second structure variable. Note that we use a period between the structure variable and member.

The output is done by means of tabs \t. You might need to accommodate the length of the entered texts to make the information be printed in the correct column. We could have done that more flexible by means of the text formatting functions from chapter 1, but we used a rough method for simplicity's sake.

9.7 Array with Structure Variables

A disadvantage with the previous program is that we needed a separate structure variable (prodOne, prodTwo) for each product. A more convenient solution is to use an array with structure variables allowing the use of a loop to process the structure variables in a uniform way.

Below we declare a structure array sProds of the type Prod with three items:

```
Prod sProds[3];
```

We have allocated memory space for three products, but we have not yet assigned values to the structure members. That could be made directly at the declaration:

```
Prod sProds[3] = {
  {"Food Oil", 101, 12.50, 100, "Felix Ltd"},
  {"Baby Oil", 102, 23.75, 25, "Baby Prod"},
  {"Boiler Oil", 103, 6100, 123000, "Shell"},
};
```

Note that the values for each structure variable are surrounded by curly brackets, and that the values are enumerated in the usual way within each pair of curly brackets. All three pair of brackets are surrounded by an extra pair of curly brackets delimiting the initiation list of values. After the last bracket there must be a semicolon.

If you want to let the user enter values, this is preferably done in a loop:

```
for (int i=0; i<3; i++)
{
  cout << "Enter values for product no. "
    << i+1 << endl;
  cout << "Start with the product name: ";
  cin.getline(sProds[i].cName, 20);
  cout << "The product id: ";
  cin >> sProds[i].iId;
  cout << "The price: ";
  cin >> sProds[i].dPrice;
  cout << "How many in stock? ";
  cin >> sProds[i].iNo;
  cin.get(); //clear in-buffer newline char
  cout << "Who supplies the product: ";
  cin.getline(sProds[i].cSupp, 25);
}
```

9.8 Pointer to Structure

You can declare pointers to structures in the same way as declaring pointers to other data types:

```
Prod *pProd;
```

To instead make the pointer point to the structure variable prodOne we write:

```
Prod* pProd = &prodOne;
```

This means that pProd equals the address to prodOne.

The advantage is to be able to use pointer arithmetics to step through the different structure variables:

```
pProd++;
```

This statement moves to the next structure variable.

With a pointer to a structure you are not allowed to use the period (.) as delimiter between the variable and member. You must use the -> characters:

```
pProd->iNo = 25;
```

For instance, to print the information for the structure pointed to by pProd, we can use the following statement:

```
cout << pProd->cName << endl <<
   pProd->iId << endl <<
   pProd->dPrice << endl <<
   pProd->iNo << endl <<
   pProd->cSupp;
```

Suppose we have declared and initiated an array sProds with space for three structure variables and with values like in the previous section:

```
Prod sProds[3] = {
  {"Food Oil", 101, 12.50, 100, "Felix Ltd"},
  {"Baby Oil", 102, 23.75, 25, "Baby Prod"},
  {"Boiler Oil", 103, 6100, 123000, "Shell"},
};
```

Then we can declare a pointer of Prod type which points to the first item of the array:

```
Prod* pProd = &sProds[0];
```

Then we can use a loop to print for example the product id:s for the three products:

```
for (int i=0; i<3; i++)
{
  cout << pProd->iId << endl;
  pProd++;
}
```

Note that we have used pointer arithmetics to step from product to product.

9.9 Structures in the Dynamic Memory

When we at the compilation cannot predict the number of products to be stored in the array, it is convenient like for other arrays (see the ‘Pointer’ chapter) put the structure array in the dynamic memory area. As usual, we accomplish this with the new keyword.

In the following code we prompt the user for the number of products to be entered, and then declare a pointer to an array in the dynamic memory space:

```
int iQty;
cout << "How many products should be
         entered? ";
cin >> iQty;
Prod *pProd = new Prod[iQty];
```

In the declaration of the array we must specify the number of structures to allocate memory for. This is done by means of the variable iQty.

Then, with a loop, we can let the user enter information to the requested number of products:

```
for (int i=0; i<iQty; i++)
{
  cin >> pProd->iId;
  // … and the remaining structure members
  pProd++;
}
```

The loop runs the number of turns as stated by the variable iQty. For each turn of the loop we increase the pointer by 1, thus making it point to the next structure variable of the array.

To print the product information we create a new loop. But first we must reset the pointer to the position of the first item of the array:

```
pProd = pProd - iQty;
for (i=0; i<iQty; i++)
{
  cout << pProd->iId << endl;
  // … and the remaining structure members
  pProd++;
}
```

Also here, we use pointer arithmetics to proceed from item to item of the array.

Finally we have to free the dynamic memory area when it will not be used any more to not block other parts of the program:

```
pProd = pProd - iQty;
delete[] pProd;
```

Innan vi använder delete, ställer vi tillbaka pekaren till sin ursprungsplats så att rätt minnesområde frigörs.

9.10 Structure As Function Parameter

9.10.1 Reference Parameter

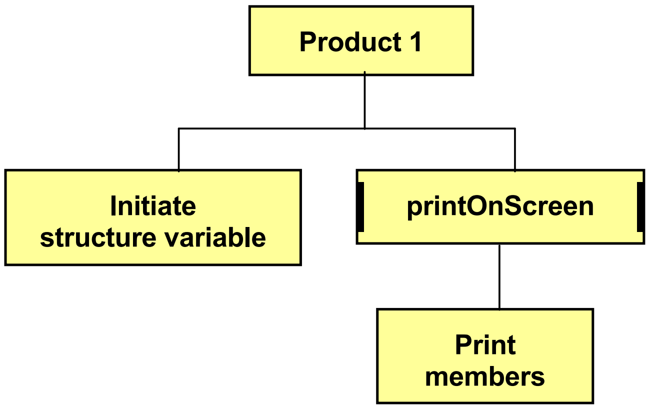
We will now show some examples of how to send structures as parameters to functions. Suppose we want to create a function which prints the content of the structure sent to the function. We give the function the name printOnScreen().

The first example of printOnScreen() takes a parameter that is a reference parameter (see the 'Functions' chapter) of a structure:

```
void printOnScreen(Prod & rProd)
{
  cout << rProd.cName << '\t' <<
    rProd.iId << "\t\t\t" <<
    rProd.dPrice << '\t' <<
    rProd.iNo << '\t' <<
    rProd.cSupp << endl << endl;
}
```

The function does not return any value (void) and takes a parameter of Prod type, which is a reference parameter (& character). It is no pointer, so we use a period as delimiter between the structure variable and member. We have used \t to tab.

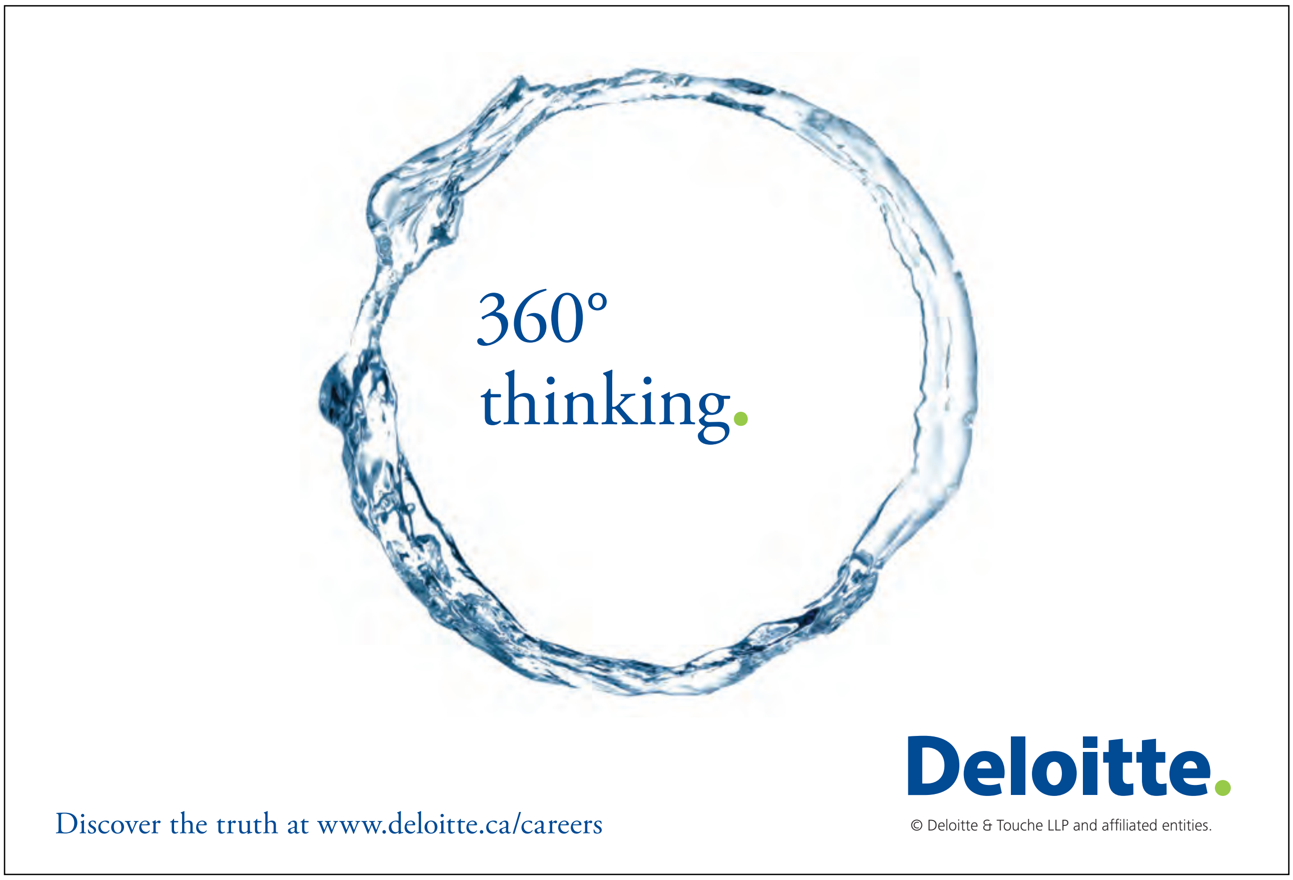
We will create a simple program to test the function. Here is the JSP graph:



We initiate a structure variable, which is sent to the function, where the members are printed.

Here is the code:

```
#include <iostream.h>
struct Prod
{
   char cName[20];
   int iId;
   double dPrice;
```


360° thinking.
Discover the truth at www.deloitte.ca/careers
Deloitte.
© Deloitte & Touche LLP and affiliated entities.

```
  int iNo;
  char cSupp[25];
};
void printOnScreen(Prod & rProd)
{
  cout << rProd.cName << '\t' <<
    rProd.iId << "\t\t\t" <<
    rProd.dPrice << '\t' <<
    rProd.iNo << '\t' <<
    rProd.cSupp << endl << endl;
}
void main()
{
  Prod prodOne = {"Olive Oil", 1001, 120.50, 250, "Frescati
    Oil S/A"};
  printOnScreen(prodOne);
}
```

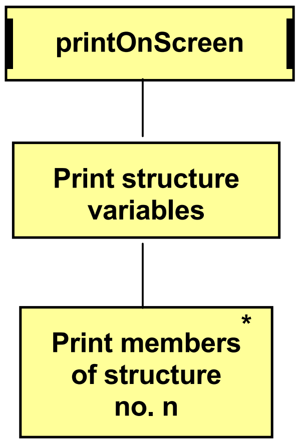
First we define the structure Prod. Then there is the function printOnScreen(). In main() we declare a structure variable prodOne of the Prod type and initiate it with values. The the function printOnScreen() is called, to which we send prodOne as actual parameter.

9.10.2 Array Parameter

Next variant of the function printOnScreen() takes a parameter which is a structure array and a parameter to the number of items in the array.

Remember that, when sending an array as a parameter to a function, it is always made on reference basis. Therefore, you should not explicitly specify that it is a reference parameter by using the & character.

Here is the JSP graph for the function:



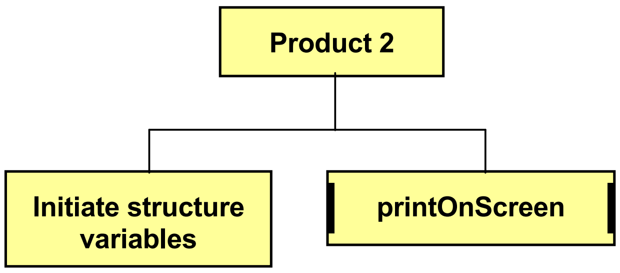
Here is the code:

```
void printOnScreen(const Prod p[],
   const int n)
{
   for (int i = 0; i < n; i++)
       cout << p[i].cName << '\t' <<
       p[i].iId << '\t' <<
       p[i].dPrice << '\t' <<
       p[i].iNo << '\t' <<
       p[i].cSupp << endl << endl;
}
```

The first parameter is of Prod type and has the name p with a subsequent square bracket to indicate array. The second parameter is called n and corresponds to the number of items of the array. Both have been given the const keyword to ensure that the information is not updated in the function.

The loop has the counter i and performs as many turns as there are items in the array. The variable i is used as index in the array to indicate specific items.

We create a simple program to test the function. The JSP graph is:



Here is the entire code:

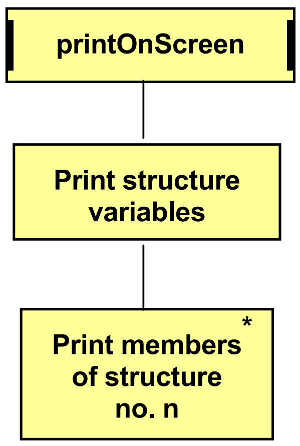
```
#include <iostream.h>
struct Prod
{
   char cName[20];
   int iId;
   double dPrice;
   int iNo;
   char cSupp[25];
};
void printOnScreen(const Prod p[],
   const int n)
{
  for (int i = 0; i < n; i++)
      cout << p[i].cName << '\t' <<
      p[i].iId << '\t' <<
      p[i].dPrice << '\t' <<
      p[i].iNo << '\t' <<
      p[i].cSupp << endl << endl;
}
void main()
{
  Prod sProds[3] = {
    {"Food Oil", 101, 12.50, 100, "Felix Ltd"},
    {"Baby Oil", 102, 23.75, 25, "Baby Prod"},
    {"Boiler Oil", 103, 6100, 123000, "Shell"},
  };
  printOnScreen(sProds, 3);
}
```

In main() we declare an array of Prod type with three items, which are initiated with values. The function printOnScreen() is called, to which we send the structure array and the number 3 as actual parameters.

9.10.3 Pointer Parameter

The last example of the function printOnScreen() takes a pointer to the Prod structure and the number of products as parameters.

Here is the JSP graph and the code:

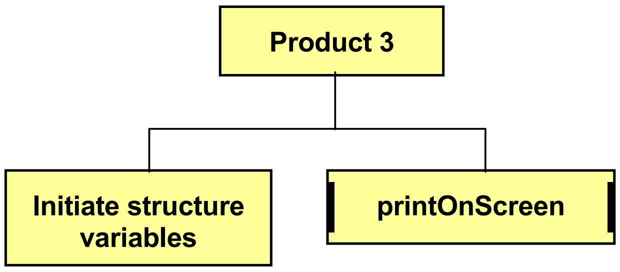


Here is the code of the function:

```
void printOnScreen(Prod *p , const int n)
{
  for (int j=0; j<n; j++)
  {
    cout << p->cName << '\t' <<
      p->iId << '\t' <<
      p->dPrice << '\t' <<
      p->iNo << '\t' <<
      p->cSupp << endl << endl;
    p++;
  }
}
```

The pointer parameter has the name p and the number of products n. The loop performs as many turns as given by the number of products. In the loop we print the members of the structure. Note that, since p is a pointer, we use -> between pointer and member. At the end of the loop we use pointer arithmetics and increase p by 1, i.e. moves p to the next structure.

We create a simple program to test the function. Here is the JSP graph:



Here is the entire code:

```
#include <iostream.h>
struct Prod
{
   char cName[20];
   int iId;
   double dPrice;
   int iNo;
   char cSupp[25];
};
void printOnScreen(Prod *p , const int n)
{
   for (int j=0; j<n; j++)
   {
      cout << p->cName << '\t' <<
         p->iId << '\t' <<
         p->dPrice << '\t' <<
         p->iNo << '\t' <<
         p->cSupp << endl << endl;
      p++;
   }
}
void main()
{
   Prod sProds[3] = {
      {"Food Oil", 101, 12.50, 100, "Felix Ltd"},
      {"Baby Oil", 102, 23.75, 25, "Baby Prod"},
      {"Boiler Oil", 103, 6100, 123000, "Shell"},
   };
   Prod *pProd = &sProds[0];
   printOnScreen(pProd, 3);
}
```

In main() we declare an array of Prod type with three items and initiate it with values. Then we declare a Prod pointer to point to the first item of the array (the & character means 'the address to'). The pointer and the number 3 is sent as parameters to the function.

9.11 Summary

In this chapter we have learnt what a structure is, namely a tool to handle items of different information that in some way belong together, for instance all data for a customer, all data for a product etc. We have learnt to define structures and declare structure variables and fill the structure with values.

We have also seen how to use arrays with structures and pointers to structures. When unable to predict the number of items to be contained by the array, we have stored it in the dynamic memory area with the new keyword.

Finally you have learnt how to send structures as parameters to functions, either as reference parameters, array parameters or pointer parameters.

You will now try your new knowledge in a number of exercises.

9.12 Exercises

1. Define a structure with customer information: name, address, customer category (one letter), discount percent, total invoice amount year-to-date. Then write a program that declares a structure variable and initiates it with some values according to your preference. Then print the information on the screen.
2. Change the previous program so that the user can enter the customer information.
3. Change the previous program to declare a structure array with 3 customers and initiates the structure members directly at the declaration. All three customers should then be printed.
4. Change the previous program so that the user can enter an order total to be added to the member 'total invoice amount year-to-date'. Then print all information.
5. Change the previous program to let the user enter information for the three customers.
6. Write a program that uses the structure definition for customers according to the previous exercises and declares a pointer to a customer. It should then be possible to enter information for the customer. The program should finally print the customer information by using the pointer.
7. Change the previous program to let the pointer point to an array with three customers. Entry and printing as before.
8. Change the previous program to let the user first specify the number of customers to be entered. The array should reside in the dynamic memory area.
9. Write a function that stores a Prod structure in a file. Use the Prod definition given previously in the chapter. Each time the function is called a new structure is written to the file without destroying previous information. Use reference parameters. Also write a main() program to test the function. Examine the file content afterwards by means of the Notepad program.
10. Change the function in the previous example to print an array of structures to the file. The array and number of items in the array should be sent as parameters.
11. Change the function in the previous example to take a pointer to the structure array and the number of products as parameters.
12. Complete the previous program with a function that reads from the file and presents the information on the screen.