

# Lab 10

## Structures (Struct) and Files

Dr. Abraham

Up until now you have been declaring variables of simple (atomic) data types. Suppose you want to keep inventory of things you sell. Each item you have in stock has a name (string), a price (double), and a quantity (integer). It would be nice to keep them together and give it one name. That is what “Struct” does for us. In other languages it may be called a “Record”. We can use a *class* as well which is covered in the next chapter. With object oriented programming it does not make sense to use struct any more. Nevertheless we C++ needs to be backward compatible to some extent. So let us dive directly into struct. To set up variables of a struct data type, one must declare a data type and then declare variables of that data type. Here is an example of declaring a struct data type. Each component of the struct is called a data members or fields. In the following example we will declare a struct data type called InvItem.

```
struct InvItem
{
    int partNum;
    string description;
    int onHand;
    float price;
};
```

InvItem in this example is used like a type. It can't be used to place values in it. A variable must be declared before using it. Here is how to declare the variable “part” and initialize it.

```
InvItem part = {1111, “Fountain Pen”, 10, 20.25}
```

Once the struct data type is declared, variables of this type can be made just as any other variable declaration. You can create any number of variables out of this struct, as in the following program 10-1. Values can be placed in the member fields either by initializing as shown above or by assigning values using the assignment operator (=) or by reading like this:

```
part.partNum = 1111;
cin >> part.partNum;
```

### Program 10-1

```
/* This program has a function that uses structure.
Created for my CSCI 1370 students
Dr. John P. Abraham */

#include <iostream>
#include <string>
using namespace std;
```

```

struct InvItem
{
    int partNum;
    string description;
    int onHand; //inventory on hand
    float price;
};

//2 prototypes
void getItem(InvItem&);
void showItem(InvItem);

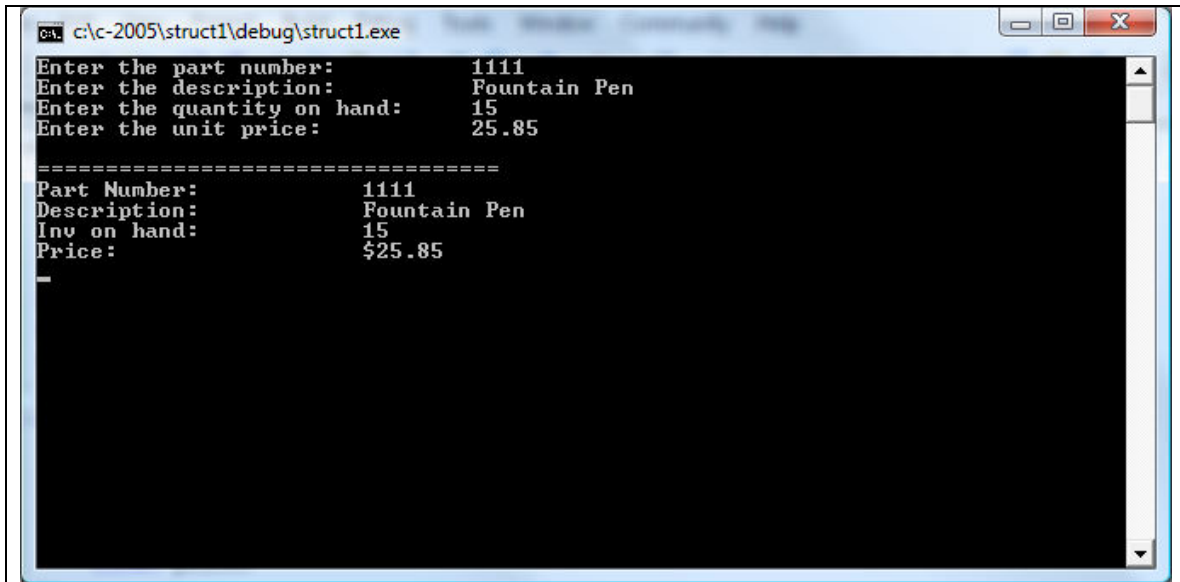
int main()
{
    InvItem part, copyPart;
    getItem(part);
    copyPart = part;
    showItem(copyPart);
    cin.ignore();getchar();
    return 0;
}

void getItem(InvItem& piece)
{
    cout<<"Enter the part number:      "; cin>>piece.partNum;
    cout<<"Enter the description:      "; cin.get();
    getline(cin,piece.description);
    cout<<"Enter the quantity on hand:  "; cin>>piece.onHand;
    cout<<"Enter the unit price:         "; cin>>piece.price;
}

void showItem (InvItem piece)
{
    cout<<"\n=====\n";
    cout<<"Part Number:          "<<piece.partNum<<endl;
    cout<<"Description:          "<<piece.description<<endl;
    cout<<"Inv on hand:          "<<piece.onHand<<endl;
    cout.precision(2);
    cout.setf(ios::fixed|ios::showpoint);
    cout<<"Price:                $"<<piece.price<<endl;
}

```

Program Run 10-1



```
ca: c:\c-2005\struct1\debug\struct1.exe
Enter the part number:      1111
Enter the description:     Fountain Pen
Enter the quantity on hand: 15
Enter the unit price:     25.85

=====
Part Number:              1111
Description:               Fountain Pen
Inv on hand:               15
Price:                     $25.85
-
```

## Files

We have already visited files in earlier chapters where we created files to create print files. After opening a file, a stream of characters were sent to the file through a buffer. Finally when a file is closed an end of file marker (^Z) is placed to indicate that the stream has ended. A file stream can be either input stream or output stream. In C++ and many other languages, peripheral devices are considered as files as well. Standard input (keyboard file) and output (monitor file) are automatically opened. All we had to use was the *cin* >> or *cout* <<. These streams are made available to us through the *iostream*. In order for us to create other streams we need to include *fstream*. This include file will allow us to use two classes: *ofstream* for output file stream and *ifstream* for input file stream. Using these classes we can create variables and throughout the program we refer to these variables we created. Each variable we create inherits member functions that can be invoked using the dot notation, example: `outfile.open ("myfile.txt")` or `outfile.close()`.

**Stream:** Channel or circuit on which data are passed from sender to receiver **one character at a time**. If receiving data, it is called **input stream**. If sending data it is called **output stream**.

**Streams** are connected to **Devices**. Example, *cin* is connected to keyboard and *cout* is connected to monitor.

**file streams** are stored on disk device.

Here some *ios* member functions:

<code>ios::in</code>	Open in input mode
<code>ios::out</code>	Open in output mode
<code>ios::app</code>	Open in append mode
<code>ios::ate</code>	Go to the end of opened file
<code>ios::binary</code>	Open in binary mode
<code>ios::trunc</code>	Delete file contents if it exists

ios::nocreate if file does not exist, open fails  
ios::noreplace if file exists, open for output fails.

C++ provides for error checking while dealing with files. For examples, what happens if a user does not put a disk in drive A? A robust program should check for errors and recover from it. C++ provides an include file, *assert.h*, to help us catch errors. In the following examples I will be using some functions provided by *assert.h*.

#### Program 10-2 (Writing to a file)

```
#include <iostream>
#include <fstream>
#include <assert.h>
using namespace std;

int main()
{
    fstream outfile ;
    outfile.open ("a:test",ios::app);

    assert(! outfile.fail());
    outfile << 25 <<" " << 38 << " " << 95 <<endl;
    outfile.close();
return 0;
}
```

#### Program Run 10-2

```
Assertion failed: ! outfile.fail(), file c:\c-2005\files1\files1\files1.cpp, line 11
```

#### Important:

Re-run the program with `outfile.open ("c:\test",ios::app);`

#### Program 10-3 (Read from a file)

```
#include <iostream>
#include <fstream>
#include <assert.h>
using namespace std;

int main()
{
    int a;
    fstream infile ;
    infile.open ("c:\test",ios::in);
```

```

        assert(! infile.fail());

/* Here are two other loops you could use
for (;;)
{
    infile >> a;
    if (infile.eof()) break;
    cout << a << endl;
}

do
{
    infile >>a;
    if (!infile.eof()) cout <<a<<endl;
}
while (infile.peek() !=EOF);
*/
while ((infile.peek() !=EOF)
{
    infile >>a;
    if (!infile.eof()) cout <<a<<endl;
}
infile.close();           //close file.
cin.ignore();getchar();
return 0;
}

```

Here is a program to make a copy of a text file.

#### Program 10-4

```

/*****
Program by Dr. John P. Abraham
Program copies a text file.
for Students of 1370
*****/
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
    char infilename[30], outfilename[30];
    cout << "Enter file name to copy from: "; cin>>infilename;
    cout << "Enter file name to copy to:   "; cin>>outfilename;
    ifstream infile; ofstream outfile;
    infile.open(infilename);
    outfile.open(outfilename);
    char inc;
    infile.get(inc);
    while(!infile.eof())
    {
        infile.get(inc);
        cout <<inc;
        outfile << inc;
    }
}

```

```
infile.close();  
outfile.close();  
return 0;  
}
```

**Assignments** (the first one is easy, the second one is challenging):

1. Re-write the file copy program (10-4) with two functions one to open the infile and another one to open the outfile. One other modification to the program would be instead of reading and writing one character at a time, read one line and write one line at a time (use `getline`). It is important to know to pass files (streams) as reference parameters.
2. Write a program to write struct in program 1 into a text file. Write another program to read that data into struct.