Writing Large Programs

You have learned many tools for writing programs. However, most of the programs so far have been fairly small. In this lesson we will discuss ways of writing larger programs.

Imagine that you have been asked to write an inventory control program for a store. The boss has given you 6 months to finish!

Do not wait 5 months and 28 days to start! You may not have all of the information to complete every detail at this point, but the boss has given you a sketch of what is required:

Inventory

In the morning when the store opens, the program will read in a file with the name of a product, the price, and the quantity in stock. During the day, some activities (transactions) occur: deliveries arrive, things are sold, some things need to be reordered. Some of these transactions affect the quantity in stock. At the end of the day, the program writes a new file with the new quantities. This file will be used to start the program the next day. (The old file will be saved as a backup.)

Part I: Array

From the description of the problem, we decide that the main information that the program must deal with is the name of a product, the price and the quantity in stock. We will start with arrays for each of those items. While we are writing the program it will be convenient to assign some initial values to the arrays. Later we can change this to read from the file, but we need to start someplace!

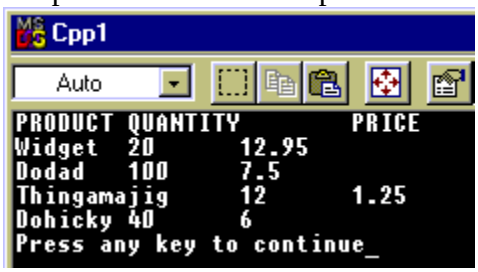
```
0 //Inventory Control
1 #include<fstream.h> //iostream is a subset of fstream.
2 void main (void)
3 { char product[20][12]={"Widget","Dodad","Thingy","Dohicky"};
4   int quantity[20] = {20, 100, 12, 40};
5   double price[20] = {12.95, 7.50, 1.25, 6.00};
6   int count=4;
7 } //main
```

As the program is being developed, the capability to print the list of all items is very important: it lets the programmer test whether the other functions are working. We will write a short program to give some initial values to the arrays and then print out the arrays.

```
 0 //Inventory Control
 1 #include<fstream.h> //iostream is a subset of fstream.
 2 void printList(char product[][12],int quantity[],
 3   double price[], int count)
 4 { cout<<"PRODUCT\tQUANTITY\tPRICE\n";
 5    for(int i=0;i<count;i++)
 6      cout<<product[i]<<"\t"<<quantity[i]<<"\t"<<price[i]
 7       <<"\n";
 8 } //printList
 9 void main (void)
10 {char product[20][12]={"Widget","Dodad","Thingy", "Dohicky"};
11   int quantity[20] = {20, 100, 12, 40};
```

```
12    double price[20] = {12.95, 7.50, 1.25, 6.00};
13    int count=4;
14    printList(product,quantity,price,count);
15  } //main
```

We notice when we run the program that the columns do not line up, and the prices do not print with 2 decimal places.



These are not serious problems, the final format of the report may change, but we leave a comment with the words TODO. Later, we can search for TODO to make sure that the little errors are cleared up.

```
0  void printList(char product[][12],int quantity[],
1     double price[], int count)
2  {  cout<<"PRODUCT\tQUANTITY\tPRICE\n";
3     for(int i=0;i<count;i++)
4       cout<<product[i]<<"\t"<<quantity[i]<<"\t"<<
5        price[i]<<"\n";
6     //TODO: line up columns and format decimal
7  } //printList
```

We would like to be able to add one item from the keyboard. We will need this eventually in order to add a new item.

Adding a fifth product to the array tests the function: *(printList is not shown here.)*

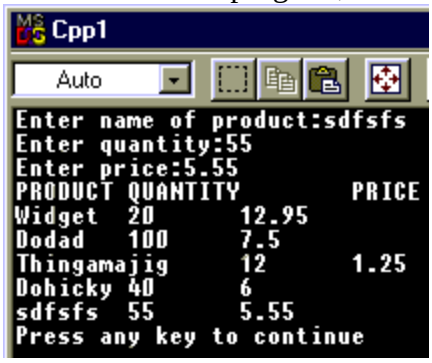
```
0  //Inventory Control
1  #include<fstream.h> //iostream is a subset of fstream.
2  #include<string.h>
3  void getItem(char prod[12], int &quan, double &price)
4  {  cout<<"Enter name of product:";
5     cin.getline(prod,12,'\n');
6     if (strlen(prod)==11) //max. letters read, get rid of \n
7        cin.ignore(100,'\n');
8     cout<<"Enter quantity:";
9     cin>>quan;
10    cout<<"Enter price:";
11    cin>>price;
12    cin.ignore(100,'\n');
13 } //getItem
14
15 void main (void)
16 { char product[20][12]={"Widget","Dodad","Thingamajig",
17       "Dohicky"};
18    int quantity[20] = {20, 100, 12, 40};
```

```
19    double price[20] = {12.95, 7.50, 1.25, 6.00};
20    getItem(product[4], quantity[4], price[4]);
21    int count=5;
22    printList(product,quantity,price,count);
23  } //main
```

We used cin.ignore after reading in a name, but discovered that sometimes we had to press enter twice after typing a name.

We test and realize that if the name is short, getline consumes the \n. It is only when the name is long that we need to ignore whatever is left in the input buffer. In order to use strlen, we include string.h.

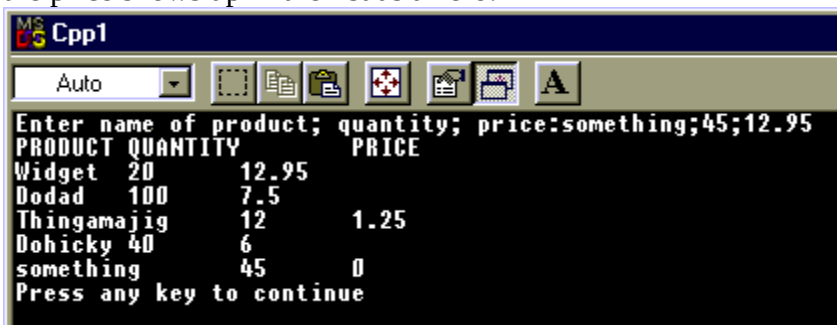
When we run the program, we can see that the fifth item has been added successfully!



We copy getItem and modify it to read all 3 items on one line, separated by semicolons

```
0  void getItem2(char prod[], int &quan, double &price)
1  {  //line of input will be: product; quantity; price\n
2     cout<<"Enter name of product; quantity; price:";
3     cin.getline(prod,12,';'); //stop at the ;
4     if (strlen(prod)==11)
5       //max. letters read, get rid of extra letters and ;
6       cin.ignore(40,';');
7     cin>>quan;
8     cin>>price;
9     cin.ignore(100,'\n'); //ignore trailing blanks and \n
10 } //getItem2
```

We can see by the output that the product and quantity were read in successfully, but the price shows up in the list as a zero:

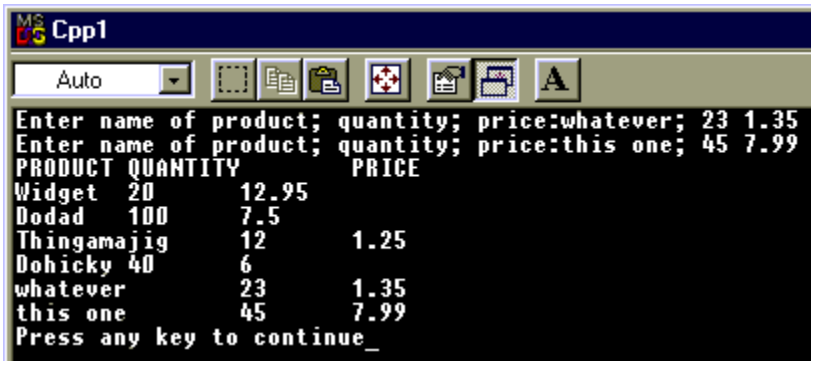


We run it one more time. This time we use a semicolon after the product, but just a space after the quantity.

White space (spaces, tabs, and line feeds) are ignored when reading numbers. This works. We have other choices: we could use cin.ignore to skip over the semicolon after reading the quantity, or we could use spaces to separate all of the fields. We would rather not use a space as the delimiter for the product, because then we could not have a space in a product name.

We are going to try reading in 2 products to see if there is a problem between reading the price on one line and trying to read the product on the next line.

```
0 void main (void)
1 { char product[20][12]={"Widget","Dodad","Thingamajig",
2    "Dohicky"};
3   int quantity[20] = {20, 100, 12, 40};
4   double price[20] = {12.95, 7.50, 1.25, 6.00};
5   getItem2(product[4], quantity[4], price[4]);
6   getItem2(product[5], quantity[5], price[5]);
7   int count=6;
8   printList(product,quantity,price,count);
9 } //main
```



.. And we like what we see! *(After mentioning the possibility of a space in a product name, we gave that a try also.)*

Part II: File

We can now create a file using the format above that we know worked when we tested it from the keyboard.

Create a file with the name of the product followed by a semicolon; then the quantity followed by a space and the price.
Save the file as A:\products.txt.

```
Gegaw; 55 1.25
Whatsit; 12 7.50
Something; 30 2.95
Widget; 100 1.25
Dodad; 0 29.95
Thingmabob; 25 9.25
```

We will start by reading just one product from the file.

The first step is to copy readItem2 and name it readFileItem. It will receive an input file stream as the forth item. Then we can change all of the cin statements to read from the file:

```
0  void getFileItem(char prod[], int &quan, double &price,
1     ifstream &input)
2  { //line of input will be: product; quantity; price\n
3    input.getline(prod,12,';'); //stop at the ;
4    if (strlen(prod)==11)
5      //maximum letters read, get rid of extra letters and ;
6      input.ignore(40,';');
7    input>>quan;
8    input>>price;
9    //ignore rest of line trailing blanks and \n
10   input.ignore(100,'\n');
11 } //getFileItem
```

We can compile it to make sure there are no syntax errors, but we can not test it without opening a file.

The function readFile receives the arrays and count from main (arrays are always passed by reference, count is passed by reference also.) The input stream file is declared and opened. Then readFile calls getFileItem, passing just one item form each array. One is added to count and the file is closed.

```
0  void readFile(char product[][12],int quantity[],
1    double price[], int &count)
2  { ifstream prod;  //input file stream
3    prod.open("A:/products.txt");
4    //always test for fail: file missing, disk not there etc.
5    if (!prod.fail())
6    { getFileItem(product[count],quantity[count],price[count],
7        prod); //read 1 item
8      count++; //add 1 to count, which was passed by reference.
9      prod.close(); //close any file you open
10   } //file opened and read
11 } //readFile
```

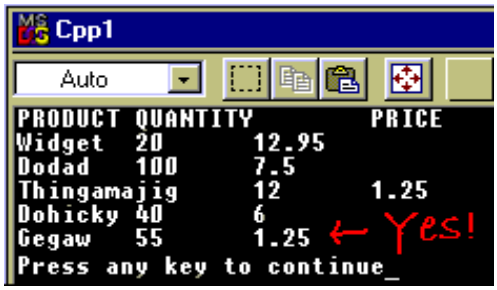
We can compile it to make sure there are no syntax errors, the next step is to call it from main.

After main is modified, we should see the first item in the file added to the bottom of the list. Notice that the calls to getItem2 are commented out. Don't delete lines that work until you are sure that the replacement works.

```
0  void main (void) //main has 3 arrays and count that are
1  passed to other functions
2  { char product[20][12]={"Widget","Dodad","Thingamajig",
3     "Dohicky"};
4    int quantity[20] = {20, 100, 12, 40};
5    double price[20] = {12.95, 7.50, 1.25, 6.00};
6    int count=4;
```

```
  //getItem2(product[4], quantity[4], price[4]);
  //getItem2(product[5], quantity[5], price[5]);
  readFile(product, quantity, price, count);
  printList(product,quantity,price,count);
} //main
```

We are now ready to run the program. We are looking for the first item from the file to appear at the end of the list. And there it is!



We can now go ahead and loop until end of file and read in all of the products. ***We should save the complete program to a backup at this point!***

```
void readFile(char product[][12],int quantity[],
  double price[], int &count)
{ ifstream prod;  //input file stream
  prod.open("A:/products.txt");
  //always test for fail: file missing, disk not there etc.
  if (!prod.fail())
  { getFileItem(product[count],quantity[count],price[count],
      prod); //read 1 item
    while(!prod.eof() && count<20)
    { count++; //add 1 to count, which was passed by ref.
      getFileItem(product[count],quantity[count],
       price[count], prod); //try to read
    } //while not end of file
    prod.close(); //close any file you open
  } //file opened and read
} //readFile
```

We are now ready to run the program. We are looking for all of the items from the file to appear at the end of the list.

Yes! They are all there!

We should save the complete program to a backup at this point!

We will now clean up the initial values from main, so that only the products read from the file appear in the list.

```
0 void main (void)
1 //main has 3 arrays, count that are passed to other functions
2 { char product[20][12];
3   int quantity[20];
4   double price[20];
5   int count=0;
6   //read in products from file
7   readFile(product, quantity, price, count);
8   printList(product,quantity,price,count);
9 } //main
```

We now have a program that reads a file of products into an array and prints them out.

We now have several choices for how to proceed.

- We can add a lookup to input a product and print out information about it;
- We can change main to a menu driven program;
- We can clean up the output for the list;
- We can add another function such as printing the reorder list.
- We can write the file back out to disk.

Writing the reorder function is pretty much a copy of printList, so we should clean up the output before we copy it.

We don't have a way to change anything in the array, so writing the file back out to disk is going to be hard to test.

Format the Output

The output list is starting to annoy us, so we decide to fix that up.

The first problem is that long names end in the second tab column. When we tab from there, we are in the third tab column. The maximum for a product name is 12 (11 letters and room for the \0). We decide print to out 15 characters for each product: if the product name has 11 letters we will print 4 spaces; ff the name has 5 letters, we will print 10 spaces. In general that is 15-the length of the name.

We first write a function to print any number of spaces. To keep it simple we will write this function and test it in a new program

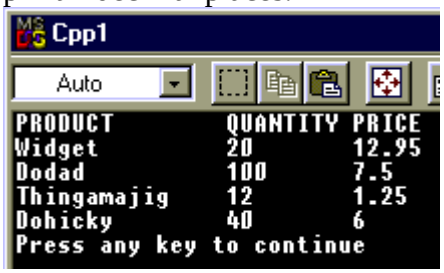
```
 0 //Test
 1 #include <fstream.h>
 2 #include <string.h>
 3 void spaces(int num)
 4 { for(int i=0; i<num; i++)
 5     cout<<" ";
 6 } //print num spaces
 7
 8 void main(void)
 9 { cout<<"ABC";
10   spaces(3);
11   cout<<"XYZ\n";
12 } //main
```

Once we know this works, we can add it to the inventory program. The inventory program is getting large. Some functions we will have no choice but to test in that program, but when we can it will be easier to test separately.

Next we go back to one of the first programs and modify printList.

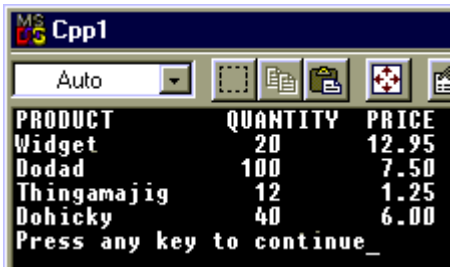
```
 0 void printList(char product[][12],int quantity[],
 1   double price[], int count)
 2 //print list of all items
 3 {  cout<<"PRODUCT        QUANTITY\tPRICE\n";
 4          //PRODUCT+8 spaces=15
 5    for(int i=0;i<count;i++)
 6    { cout<<product[i];
 7      spaces(15-strlen(product[i])); //15- length of name
 8      cout<<quantity[i]<<"\t"<<price[i]<<"\n";
 9    } //for i
10    //TODO: line up columns and format decimal
11 } //printList
```

The product names line up, but the quantity is not right justified and the price does not print 2 decimal places.



In order to format the output, we need to include iomanip.h

```
0  void printList(char product[][12],int quantity[],
1     double price[], int count)
2  //print list of all items
3  {  cout<<"PRODUCT        QUANTITY\t PRICE\n";
4           //PRODUCT+8 spaces=15
5     for(int i=0;i<count;i++)
6     { cout<<product[i];
7       spaces(15-strlen(product[i])); //15- length of name
8       cout.precision(0);
9       cout.setf(ios::fixed);
10      cout<<setw(4)<<quantity[i]; //xxxx = 4
11      cout.precision(2); //2 decimal places
12      cout<<"\t"<<setw(6)<<price[i]<<"\n"; //xxx.xx =6
13    } //for i
14 } //printList
```



Ok, good, it works!

We can add the code for printList, spaces, and include iomanip.h to the inventory program. **This is a good time to save!**

At this point, the program reads a file into 3 arrays and can print a list of all the items. The output looks pretty much the way we think it should look. Let's take a look at our original list:

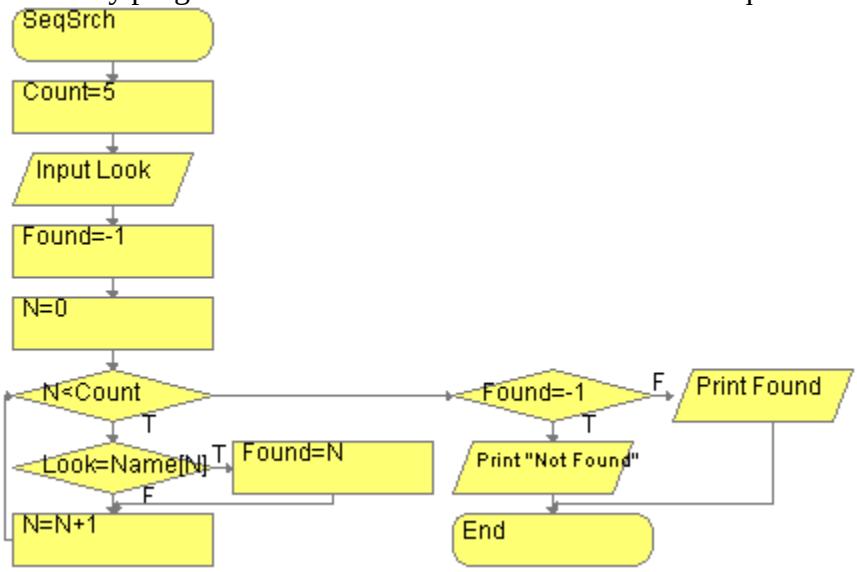
The program should read the product information into arrays (DONE), then allow the user to select any of the following using a menu driven program.

- Print a list of all items, prices and quantity; (DONE)
- Check the price or availability of an item;
- Sell an item;
- Receive delivery of an item;
- Print a list of items to be reordered;
- Quit (at which time the file is saved.)

Several items on the list involve finding one item and displaying it, or modifying it. For this to work we need a search.

Sequential Search

We will go back to the original program to test the search before we add it to the inventory program. Please look at the flowchart for the sequential search:




```
0  //Test search
1  #include<fstream.h> //iostream is a subset of fstream.
2  #include<string.h>
3
4  int search( char lookfor[], char product[][12],int count)
5  //returns position in array, or -1 if not found
6  { int found=-1;
7    for(int i=0; i<count; i++)
8      if(strcmp(product[i], lookfor)==0)
9         found=i;
10   return found;
11 } //search
12
13 void main (void)
14 //main has 1 array, count that are passed to other functions
15 { char product[20][12]={"Widget","Dodad","Thingamajig",
16    "Dohicky"};
17   int count=4;
18   cout<<search("Dohicky",product,count);
19   //try other words, including not on list
20 } //main
```

The main problem with this search is that it does not stop looking after it finds what it is looking for. The search is modified to stop looking after we find what we are looking for.

```
0  int search( char lookfor[], char product[][12],int count)
1  //returns position in array, or -1 if not found
2  { int found=-1;
3    int i=0;
4    while(i<count && found==-1)
```

```
  { if(strcmp(product[i], lookfor)==0)
       found=i;
    i++;
  } //while loop
  return found;
} //search
```

Next we will add a function to ask the user the name he is looking for.

```
//Test search
#include<fstream.h> //iostream is a subset of fstream.
#include<string.h>

int search( char lookfor[], char product[][12],int count)
//returns position in array, or -1 if not found
{ int found=-1;
  int i=0;
  while(i<count && found==-1)
  { if(strcmp(product[i], lookfor)==0)
       found=i;
    i++;
  } //while loop
  return found;
} //search
void lookup(char product[][12],int quantity[],double price[],
   int count)
{ char lookfor[12];
  cout<<"Enter the product name:";
  cin.getline(lookfor,12,'\n');
  int position=search(lookfor,product,count);
  if (position==-1)
    cout<<"Sorry, not found\n";
  else
    cout<<quantity[position]<<"\t"<<price[position]<<"\n";
} //lookup

void main (void)
//main has 1 array, count that are passed to other functions
{ char product[20][12]={"Widget","Dodad","Thingamajig",
   "Dohicky"};
  int quantity[20] = {20, 100, 12, 40};
  double price[20] = {12.95, 7.50, 1.25, 6.00};
  int count=4;
  lookup(product,quantity,price,count);
} //main
```

This works, so we will add this to the inventory program. Of course, we save again!

Menu Driven

Our program is getting rather large. We are ready to make this a menu driven program. Then we can add and test individual functions one by one. *(Please refer to the lesson on menu driven programs. We will refer to several functions from the shape program.)*

Copy and paste the function **wait** (*waits for the user to press enter before continuing*) from the Shape program into the inventory program.

We will modify the menu function as shown below. We can compile and check for syntax errors but we are not ready to test.

```
 0 void menu(void) //display the choices
 1 { cout<<"Inventory Control\n\n";
 2   cout<<"Enter the letter of your choice: \n";
 3   cout<<"A: Add new product\n";
 4   cout<<"B: Buy an item\n";
 5   cout<<"C: Add to quantity of an item\n";
 6   cout<<"D: Delete a product\n";
 7   cout<<"I: Inquire about a product\n";
 8   cout<<"P: Print list of all products\n";
 9   cout<<"R: Print reorder list\n";
10   cout<<"Q: Quit\n";
11   cout<<"Please enter your choice:";
12 } //menu
```

Next we will copy and paste the function doChoice. We have to pass all three arrays and count to doChoice so that it can pass them on to the rest of the functions.
We will modify the doChoice function as shown below.

```
 0 void doChoice(char choice,char product[][12],int quantity[],
 1 double price[], int &count) //calls function selected
 2 { switch (choice)
 3   {case 'A': case 'a':
 4      addProduct(product, quantity, price, count); break;
 5    case 'B': case 'b':
 6      buyProduct(product, quantity, price, count); break;
 7    case 'C': case 'c':
 8      increaseQuantity(product, quantity, price, count);
 9      break;
10    case 'D': case 'd':
11      deleteProduct(product, quantity, price, count);
12      break;
13    case 'I': case 'i':
14      lookup(product, quantity, price, count); break;
15    case 'P': case 'p':
16      printList(product, quantity, price, count); break;
17    case 'R': case 'r':
18      reorderList(product, quantity, price, count); break;
19   } //choice
20   wait();  //let user look at result before menu displayed
21 } //doChoice
```

If we compile, we will get error messages for all of the functions that do not exist.

Stub Functions

There are two ways to deal with this: comment out all of the functions that don't exist or write "stub" functions. A "stub" function is one with the correct format, but no code: usually we print some message giving the name of the function so that we can test the rest of the program. Since we have been using TODO to search for later, we will include that again.

```
0 void buyProduct(char product[][12],int quantity[],
1   double price[], int count)
2 { cout<<"TODO: write addProduct\n";
3 } //addProduct
```

We will add identical "stub" functions for addProduct, buyProduct, increaseQuantity, deleteProduct, and reorderList.

Function main is now modified to implement the menu:

```
 0 void main (void)
 1 //main has 3 arrays, count that are passed to other functions
 2 { char product[20][12];
 3   int quantity[20];
 4   double price[20];
 5   int count=0;
 6   char choice;
 7   readFile(product,quantity,price,count);
 8   do
 9   { menu();            //display choices
10     cin>>choice;
11     cin.ignore(100,'\n');
12     doChoice(choice,   product,   quantity,   price,   count);
13 //calls function selected
14   } while ((choice != 'Q') && (choice !='q'));
15   cout<<"TODO: save file!\n";
16 } //main
```

There are many more finishing touches that we will add, but we are well on the way!