
# TRIBHUVAN UNIVERSITY

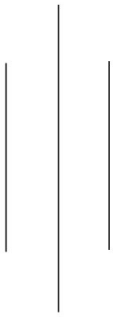
# INSTITUTE OF ENGINEERING

# PULCHOWK CAMPUS

A COURSE PROJECT ON

## SUPERMARKET BILLING SYSTEM

SUBMITTED TO DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE PRACTICAL COURSE ON
COMPUTER PROGRAMMING



**SUBMITTED TO:**

Department of Electronics and computer engineering, Pulchowk Campus

Institute of Engineering, Tribhuwan University

Lalitpur, Nepal

**SUBMITTED BY:**

Bigyan Adhikari (080bel021)

Bikalpa Bhattarai (080bel023)

Anish Khadka (080bel013)

# ABSTRACT

There are many uses of a computer. People use computer according to their requirements. The lifestyle of people has also changed greatly because of the advancements in computer world. The daily job of people re also made easier and efficient by personal computers. People can perform various tasks online to make their life easy. Modern world cannot imagine proper development without the use of computers.

Various useful applications and software can be developed in a PC using a programming language. A programming language is a formal language that specifies a set of instructions that can be used to solve problems of the users. It is the language used by the programmers to communicate with the computer and instruct it to solve the problem of the users. Some of the programming languages are: C, C++, C#, Java, Python, QBASIC, etc. Out of all these programming languages, we are primarily focused on C-programming language. C-programming language was developed by Dennis M. Ritchie and is widely used to develop several system and application software. There is a wide range of application for this programming language.

This is a mini project on topic "SUPERMARKET BILLING SYSTEM" which is about storing the information or details of each products that are going to be sold in the supermarket. Also this program helps to update, search and view overall particulars being sold in the store. The main function of the program is to create a bills according to customer requirements and help the user on counter with easier calculations of the bills and give a basic but detailed information of products stored in file in computer of can be said as in stock of supermarket.

# ACKNOWLEDGEMENTS

First of all, We would like to express our deep gratitude towards our computer teacher, **Mrs. Naina Amatiya** who gave us the concept of C programming language. Also we would like to thank all our dear friends and seniors for their advice, encouragement, support and help in the process of bringing the work into present form

# TABLE OF CONTENTS

# LIST OF FGURES AND ABBREVIATIONS

## Abbreviations

- UDF → User Defined Function
- PC → Personal Computer
- OS → Operating System
- IOE → Institute of Engineering

## List of figures

*Chapter one*

# INTRODUCTION

For those pursuing technical education in any field, the most important requirement to show their ability and skill is the practical performance rather than their theoretical understanding. Based on it this project naming "SUPERMARKET BILLING SYSTEM" is prepared for better understanding in the field of programming.

## 1.1 Background of the project

This project was prepared using c-programming language using simple logic available in high level language of this programming language. This program concerns with recording and maintaining the data about the products being sold in super-market. Along with that it also helps the user at the counter to make bills with minimal effort and time that increases the efficiency of the super-market. This program helps for not only making the bill but also helps to calculate total amount and adds required discount or tax as reqd. for the product. This program used a lot of functions of c programming which also includes array, structures, file handling and user defined functions for calculation and storing of data files for use in the program.

## 1.2 Statement of the problem

We have usually seen in the general stores or any store that sell products make handwritten paper bills that is quite time consuming to make and requires lots of technical knowledge of calculations to make a correct bill, which doesn't remove human factor that is never perfect. Also, when products are stored in stock, they usually do not have their records written which doesn't let the store operator to have perfect knowledge of their stocks of product, and maintaining it in hand written form is not feasible. Even, to know basic details of products we need to go to the store room regularly.

These all being said are quite hard and time consuming, which might not effect a small stores having limited amount of products to sell but it is not the case for supermarket. Supermarkets sell thousands rather say hundreds of thousand products and keeping their record in written form is possible. And the second but main problem is the customers, supermarkets have very large number of customer visiting it all the time and making their bills is not possible by the old handwritten bills.

## 1.3 Objective of the project

For all the problems discussed above, we need a strong method that is error proof and less time consuming which increases the efficiency of the supermarket. This

project being made has two different categories of objectives that help us to learn more about the computer programming and helping supermarket operators in that very problem of making bills and recording products details for less time consumption and less errors during calculations.

Its academic objectives for us are:

- To learn about different library functions included in different header files.
- To learn about the use of user defined function, structure, array and data file in C.
- To learn to be able to develop complex programs aimed at solving particular task in practical field as per users requirements.
- To be able to work in group as a team sharing different responsibilities and improving team spirit.

The objectives of the project in problem solving field are:

- To record all the details of products in computer memory.
- To do all most of the calculation works in miliseconds without errors.
- To help operator analyze the stocks in the store.
- To create bills of customer that may include a single product or hundreds of products in very less time.

## 1.4 Limitations of the project

As the proposed project is an academic work, there was limited time and resource factors. Soo the project was restricted in the following areas:

- Only CUI is used in the program that might not be very attractive on the look of the operating program.
- Computers with 16-bit processing capacity or less would not be able to run this program.
- The name of product can't have spaces between them.
- Once a procedure is started in the program, it can't be returned back without completing the task, even if any errors are made in the process.

*Chapter two*

# REVIEW OF RELATER LITERATURES

This project is based on high level language i.e. c programming. **C** is a general-purpose computer programming language supporting structured programming, lexical variable scope, and recursion, with a static type system. By design, C provides constructs that map efficiently to typical machine instructions. C is structured programming based computer programming language was developed by Dennis Ritchie at Bell laboratories in 1972.Structured programming refers to programming that produce program with clean flow, clear and a degree of modularity or hierarchical structure is a simple, contained, versatile, excellent, efficient, fast general purpose language. It has high degree of language of C is a function oriented additional task including input and output, graphics, math computation and access to peripheral devices are placed as library function.

C-programming is similar to other programming language that have different types of instructions and their syntaxes that help to create programs. These include control statements, structures, array, file handling, etc. Some of the instruction types are discussed herein with their purpose, need, syntax and importance.

## PROCESSOR DIRECTIVES

Processor directives are lines included in a program that begin with the character #, which make them different from a typical source code text. They are invoked by the compiler to process some program before compilation. Preprocessor directives change the text of the source code and the result is a new source code without these directives.

Example: **#<stdio.h>**

This type of preprocessor directive is used to include a header file **stdio.h** in the program.

## INPUT AND OUTPUT

getchar() and gets() are unformatted input statement that directly store the entered data while putchar() and puts() are similar to above rather they give direct output.

Formatted input and output can also be used in C programming using the statements scanf() and printf() respectively. These help to take input and output in desired form and makes the look of the program more promising.

**scanf()**

This function takes input of all kinds of data types in user required form.

Syntax: scanf("Control string",&arg1,&arg2,……..&arg n);

**printf()**

This function displays all kind of data types according to the format specified by the user in the source codes.

Syntax: printf("Message + Control strings",&arg1,&arg2,……..&arg n);

where either message or control string with arguments or both can be used.

## ARRAY

An array in C is a collection of similar types of data, stored at contiguous memory locations. Elements can be accessed randomly using indices of an array and they are used to store similar type of elements as in the data type must be the same for all elements.

**Array declaration**

Syntax: data_type array_name[array size];

Example: float mark[10];

## Multidimensional array

The array having two or more than two indices is known as multidimensional array.

Declaring a two dimensional array

Syntax: data_type array_name[x][y];

## STRINGS AND STRING MANIPULATION

In C programming, a string is a sequence of characters terminated with a null character '\0'.The syntax for declaring a string is:

char string_variable_name [size];

## String handling functions

Following are the commonly used string handling functions used in C programming language, which are defined under "string.h" header file.

**strcpy()**

Syntax: strcpy(string1, string2);

It copies contents of string2 into string1.

**strcat()**

It concatenates two string, string1 and string2 and stores the concatenated string on string1.

Syntax: strcat(string1,string2);

**strlen()**

It computes the length of the string „string1" and returns its length to the variable "integer_variable".

Syntax: integer_variable = strlen(string1);

**strrev()**

It reverses the value of string1and keeps the reversed stringin string1 itself.

Syntax: strrev(string1);

**strcmp()**

Returns 0 if string1 and string2 are the same

Returns -1 if string1 < string2

Returns +1 if string1 > string2

Syntax: strcmp(string1,string2);

## STRUCTURE

Structure is a collection of dissimilar types of data. Use of structure in programs makes the programs morereadable and efficient.

The syntax for defining a structure is as follows:-

```
struct structure_name
{
    Data_type variable_name 1;
    Data_type variable_name 2;
    Data_type variable_name 3;
    .....
    Data_type variable_name n;
};
```

After a structure is defined, it can be declared in any function as follows:

struct structure_name variable_name;

**Accessing Structure Elements**

Each element of structure can be accessed by using dot (.) operator. If we have defined a structure called student and have declared it as follows:

struct student s;

Each element of s can be accessed by using dot operator as follows:

s.name, s.roll, s.address etc.

Input and output of element can be done by the following way.

scanf("%s", &s.name);

printf("%s", s.name);

**Array of Structure**

We can create an array of structure like we created an array of basic types.

For example:

struct student s[40];

**Structures and Functions**

Structures can be passed to function similar to passing basic types to the function.

**Structure within structure or Nested Structure**

When a structure has a member of another type of structure, it is called nested structure.

**CONTROL STATEMENTS**

Logical operation is carried out by several symmetrical or logical statements.
There are three types of control statement based on their function:

**[A] Sequential control**

In sequential control statement, the program executes the instructions in the sequential order in which they are contained in the source code. This can be called as simple flow of the program.

**[B] Selective structure**

Selective structures are used when we have a number of situations where we need to change the order of execution of statements based on certain condition. The selective statements make a decision to take the right path before changing the order of execution. C provides the following two of the statements for selective structure:

**If statements**

The if statement is a powerful decision making statement and it is used to control the flow of execution of statements. It is a two way statement and is used in conjunction with an expression If statement allows the computer to evaluate the expression first and then on depending whether the value of the expression is true or false it transfer the control to the particular statement. At this point of the program has two paths to follow: one for true condition and other for false condition. The types of if statements are explained below:

**I. Simple if statement**

The simple if statement is used to conditionally excite a block of code based on whether a test condition is true or false. If the condition is true the block of code is executed, otherwise it is skipped. The syntax of if statement is given below:

```
if(test expression)
{
    statement-block for true expression;
}
```

**II. If else statement**

The if else statement extends the idea of the if statement by specifying another section of code that should be executed only if the condition is false i.e. conditional branching. True- block statements are to be executed only if the test expression is true and false block statements to be executed only if the condition is false. The syntax of if else statement is given below:

```
if(test expression)
{
    true block statement;
}
else
{
    false block statement;
}
```

### III. Nested If else or If else ladder

This statement is used for multiple comparision and option scenario.

Syntax:

```
if(condition-1)
{
statement-1:
}
else if(condition-2)
{
statement-2:
}
……..
……..
else
{
default statement:
}
```

## [C] Looping

Loop caused a section of code to be repeated for a specified number of times or until some condition holds true. When a condition becomes false, the loop terminates and control passes to statement below loop. Different types of loops are discussed below with their major characteristics and syntax used in C:

### I. while loop

The "while loop" specifies that a section of code should be executed while a certain condition holds true. The syntax of while loop is given below:

```
while(test expression)
{
      body of loop( statements block)
 }
```

### II. do while statement

The do while statement is very similar to while statement. It also specifies that a section of code should be executed while a certain condition holds true. The difference between while and do while loop is that while loop test its condition at the top of its loop but do while loop tests its condition at the bottom of loop. In while loop, if the test

condition is false, the block of code is skipped. Since condition is tested at the bottom of loop in do while loop, its block of code is always executed at least once. The syntax of do while loop is given below:

```
do
{
body of loop
}while (test expression);
```

### III. For loop

The for loop is used to execute a block of code for a fixed number of repetitions. Initialization is generally an assignment statement used to set loop control variable. Test expression is a relational expression that determines when loop exits. Update expression defines how the loop variable changes each time when the loop is repeated. The syntax of for loop is given below:

```
for(initialization expression; test expression; update expression)
{
     body of loop;
 }
```

## [D] Goto

Goto is also a control statement which passes the flow of execution from one part of the program to another part without any condition or restriction. To use goto statement we have to create a point where goto can return the execution of the program which is termed as label.

Syntax for label: Label_ name:

Syntax for goto: goto label_ name;

## Break statement

The break statement is used to jump out of loop. The break statement terminates the execution of the nearest enclosing loop. Control passes to the statement that follows the terminated statement in a switch break statement causes the program to execute the next statement after switch.

Syntax: break;

## USER DEFINED FUNCTIONS

### Function

A function is a group of statements that together perform a task. Every C program consists of a main function and other user defined functions. The execution of a program starts from the main function and other functions can be called upon the need.

The function contains the set of programming statements enclosed by { }. A function can be called multiple times to provide reusability and modularity to the C program.

C functions are broadly classified into two major categories, they are:

i. **Library Functions**
Library functions are those type of functions which are already defined, compiled and stored in different header file of standard C library such as scanf(), printf(), gets() etc.

ii. **User Defined Functions**
Those functions which are defined by programmers according to their need are known as User Defined Functions.

Major advantages of User defined functions:

- It helps in reduction of program size.
- It helps by making easy in debugging.
- It makes program more organized and clear to understand.

### Different Components of a function:

A function consists of different components in the program. They are as follows:

**I) Function Declaration or Function Prototype**

A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type. The function prototype informs the compiler about the return type of function and argument it takes.

Syntax:

Return_type function_name(type1 arg1, type 2 arg2,....., type n argn);

## II) Function Call

The function call transfers the control of the program from the calling function to the called function.

Syntax:

Function_name(arg1,arg2,.....,argn);

## III) Function Definition

In C programming, a user defined function must be defined before it is called or used in the program. Function definition consists all the code required for its implementation.

Syntax:

return_type function_name(type1 arg1, type2 arg 2, ...., type n arg n)

## Types of User-defined Functions in C Programming

i. **Function with return type and arguments**
   This type of function returns a value as well as takes some arguments.

ii. **Function with no return type and no arguments**
   This type of function does not return any value and take no arguments.

iii. **Function with no return type and argument**
   This type of function does not return any value but takes some arguments.

iii. **Function with return type and no argument**
   This type of function does return value but do not take any argument.

## Passing arguments to Function

### Pass by Value (or Call by Value)

It is a method of function call in which the value of arguments is passed to the called function from calling function. The values of arguments in the function call are copied to the argument variables declared in the function definition of called function. Thus, in pass by value, the arguments in the function definition of called function are simply the copy (or duplicate) of arguments in the calling function.

### Pass by Reference (or Call by Reference)

It is a method of function call in which the memory of arguments is passed to the called function from calling function. The memory location of arguments in the

function call is copied to the respective pointer argument variables declared in the function definition of called function. Thus,in pass by reference, the arguments in the function definition of called function are simply the pointer variable which holds memory address of the corresponding arguments in the calling function.

## Recursive Function

The function which calls itself until a pre-defined condition is true is known as recursive function.

## FILE HANDLING IN C

During programming, it is often required that we need to work with different types of file. C programming supports file handling and allows us to read and write files programmatically.

### Concept of Data Files

We frequently use files for storing information which can be processed by our programs. Such files are called data files. A file is a bunch of bytes stored on some storage device.

### File Operations

In every file I/O, one has to do three basic operations. They are as follows:

- Opening a file.
- Performing read, write, and append operation on an opened file.
- Closing a file.

### File Pointer

We can have many files on our disk. While dealing with files, in our program, we need to specify which files we wish to use. In c programming, we use a new data type called file pointer to communicate with files. A file pointer can be declared as follows:

```
FILE *file _ pointer;
```

### Opening and closing a file

The file pointer can be used to open and close the file as follows:

```
file_pointer = fopen (“path_ string_ for _file”, “mode _string”);
fclose(file_ pointer);
```

### String IO operation in Files

In C programming language, string IO operations can be performed on a file using functions

like fgets() and fputs() using following syntax:

fputs(string_ variable ,file_ pointer);

The function fputs() writes the contents of string variable into the file pointed by file pointer.

fgets(string_ variable ,value ,file_ pointer);

The function fgets() reads strings from the file pointed by file_ pointer and copies it to the string_variable. The value represents an integer value which is the number of characters to be read from the file which is to be copied into the string_ variable.

## Formatted IO operations in Files

In C programming language, formatted IO operations can be performed on a file using functions like fprintf() and fscanf() using following syntax:

fprintf(file_ pointer, "control_ string", list_ of_ arguments);

The function fprintf() writes the value of arguments to the file being pointed by the file_ pointer under the control of control_ string.

fscanf(file_ pointer, "control_ string" ,&list_ of_ arguments);

The function fscanf() reads the formatted values from a file being pointed by the file_pointer into the list of arguments under the control of control_ string.

## Modes of opening a file

While opening the file, we need to specify the purpose for what the file is being opened using a mode string. There are lots of mode in which the file can be opened in c programming. Some of the mode strings, their file opening mode and what they perform are as follows:

**r** → Open text file for reading only. The file must already exist.

**w** → Open text file for writing only. If the file specified already exists, its contents will be destroyed. If it does not exist it will be created.

**a** → Open text file for appending (i.e. adding data to the end of existing file). If it does not exist it will be created.

**r+** → Open text file for both reading and writing. The file must already exist.

**w+** →Open text file for both reading and writing. If the file exists, its content will be overwritten. If it does not exist it will be created.

**a+** → Open text file for both reading and appending. If a file does not exist it will be created.

Different binary file modes in C programming are tabulated below:

**rb** → Open binary file for reading only. The file must already exist.

**wb** →Open binary file for writing only. If the file specified already exists, its contents will be destroyed. If it does not exist it will be created.

**ab** → Open binary file for appending (i.e. adding data to the end of existing file). If it does not exist it will be created.

**rb+** → Open binary file for both reading and writing. The file must already exist.

**wb+** → Open binary file for both reading and writing. If the file exists, its content will be overwritten. If it does not exist it will be created.

**ab+** → Open binary file for both reading and appending. If a file does not exist it will be created.

## Difference between Text File & Binary File

Text file and binary files are extensively used in programming. In this article we are going to point out the major difference between text file and binary file.

i. In text file, text, character, numbers are stored one character per byte i.e. 32667 occupies 5 bytes even though it occupies 2 bytes in memory.
In binary file data is stored in binary format and each data would occupy the same number of bytes on disks as it occupies in memory.

ii. In the text file, the newline character is converted to carriage-return/linefeed before being written to the disk.
In binary file, conversion of newline to carriage-return and linefeed does not take place.

iii. Text files are used to store data more user friendly.
Binary files are used to store data more compactly.

iv. In the text file, a special character whose ASCII value is 26 inserted after the last character to mark the end of file.
In the binary file no such character is present. Files keep track of the end of the file from the number of characters present.

v. Content written in text files is human readable.
Content written in binary files is not human readable and looks like encrypted content.

**End of File (EOF)**

EOF is a MACRO based integer value declared in stdio.h header file. When the end of the file is detected by the operating system, it automatically transmits EOF signal to indicate the end of the file. An attempt to read file after EOF signal might result in an error or infinite loop condition. The value of EOF in GNU GCC based compiler is -1. The hex value of EOF is 1A and decimal value is 26.

*Chapter three*

# ALGORITHM AND FLOWCHART

An algorithm is defined as a set of ordered steps or procedures necessary to solve a problem. Following are the properties of excellent algorithm.

- **Finiteness:** It must be finite.
- **Definiteness:** Steps in algorithm must be clearly defined.
- **Inputs:** The algorithm must take 0 or more input
- **Outputs:** The algorithm must produce one or more outputs.
- **Effectiveness:** Every instruction must be basic enough to be carried out theoretically or by using paper and pencil.

The algorithms of the developed program along with algorithms of the user defined functions used in the program are as follows:

**[a] Algorithm of Main function**

Step 1: start

Step 2: Declare sel as integer

Step 3: Call UDF nameofstore

Step 4: sel ← mainoptions()

Step 5: If sel = 1, goto UDF billmaking

Step 6: If sel=2, goto UDF productsearch

Step 7: If sel = 3, goto UDF productupdate

Step 8: If se = 4, goto UDF newproduct

Step 9: If sel = 5, goto UDF displayproduct

Step 10: If sel = 6, goto UDF instructions

Step 11: If sel = 7, Exit the program

Step 12: Else display 'invalid input' and goto step 2

Step 13: Exit

**[b] Algorithm for UDF mainoptions()**

Step 1:Start

Step 2:Declare a1

Step 3:Display to choose from following options

Display 1) Make a bill

Display 2) Search product by name

Display 3) Update product to stock

Display 4) Add new product to stock

Display 5) Display total stock information

Display 6) Instructions

Display 7) Exit the program

Step 4: a1 ← Read value of chosen option

Step 5: Return a1

Step 6:Exit

**[c] Algorithm for UDF billmakiing()**

Step 1: Start

Step 2: Declare m=0, billno as integer

Step 4: Open file "billno.txt" in read mode at file pointer fb

Step 5: If fb = NULL

billno ← 10000

else

billno ← Read from file

Close file fb

Step 6: Declare cname, caddress, opt4, opt5, opt6 as string

Step 7: cname ← Read name of customer

Step 8: caddres ← Read address of customer

Step 9: Decclare b as structure billing

Step 10: b[i].code ← read product code of a product

Step 11:b[i].quantity ← Read quantity of the product

Step 12: opt4 ← Read value for 'Add more products(y/n)?'

Step 13: If opt4 = y,

i++

goto step 14

Step 14: Declare j = 0, z = 0, k, i = lastproductcode(1), pcode as integer

Step 15: Declare ntotal=0, gtotal=0, discount, paid as float

Step 16: Declare p as structure productlist and activate database using UDF filetostructure

Step 17: Equate b[i].code with p[k].pcode

If true, goto step18, else repeat step 17 with increased value of i, k

Step 18: Read all details of product from file and store in structure 'billing'

Step 19: p[k].pamt ← p[k].pamt – b[i].quantity

Step 20: Goto step 17 until end of data in structure 'billing'

Step 21: Open file "stock.txt" in write mode at file pointer fb

Step 22: Save the data from productlist to file 'stock.txt'

Step 23: Close file fb

Step 24: discount ← Read amount of discount from user

Step 23: gtotal ← (ntotal - discount)*1.13

Step 24:Display: total amount to be paid 'gtotal'

Step 25: paid ← Read amount paid by customer

Step 26: change ← amount paid - gtotal

Step 27: Get keypress for displaying the bill

Step 28; Display details of store in formatted output

Step 29: Display name and address of customer

Step 30: Display headings for product name, Quantity, net rate, net amount

Step 31: Display details of one product from 'billing'

Step 32: Goto step 31 until end of array

Step 33: Display net total 'ntotal'

Step 34: Display discount

Step 35: Display vat percent

Step 36: Display grand total 'gtotal'

Step 37: Display Thank you! Visit again

Step 41: Open file "billno.txt" in write mod at file pointer fb

Step 42: Save billno to file

Step 43: Close file fb

Step 44: Read value for 'Make another bill(y/n)?'

if yes goto step 2

Step 45: Exit

**[d] Algorithm for UDF productsearch()**

Step 1: Start

Step 2: Declare j = 0, z = 0, i = lastproductcode(1) as integers

Step 3: Declare pname, opt4 as string

Step 4: Declare structure 'productlist' and activate database using UDF 'filetostructure()'

Step 5: pname ← Read name of product being searched

Step 6: Equate p[z].pname to pname

If TRUE, goto step 7

Else goto step 8

Step 7: Display all details of product lying in p[z]

j++

Step 8: If j=0 then display 'Product not found'

Step 9: opt4 ← Read value for : Search another product(y/n)?

Step 10: If opt4 equals to 'y'

j=0

goto label4

Step 11: Exit

**[e] Algorithm for UDF productupdate()**

Step 1: Start

Step 2: Declare file pointer fp

Step 3: Declare variable j = 0, z, k, i, pcode, update as integers

Step 4: Declare pname, opt4 as string

Step 5: Declare structure productlist

Step 6: Call UDF filetostructure()

Step 7: pcode ← Read the product code to be updated

Step 8: If p[i].pcode = pcode

display product details

j++

Else

i++ and Repeat step 8

Step 9: if j = 0

Display 'product not found'

go to step 14
Else
update ← Read no. of stocks to update
Step 10: Open 'stock.txt' in write mode at file pointer fp
Step 11: p.pcode ← p.pcode + update
Step 12: Store the value in file
Step 13: Close file fp
Step 14: Read value for: 'Update another product(y/n)?'
if value = y
go to step 4
step 15: Exit

**[f] Algorithm for UDF newproduct()**

Step 1: Start

Step 2: Declare pname, pcat, opt1, opt2 as string

Step 3: Declare pcode, pamt, I as integers

Step 4: If file 'stock.txt' is NULL

pcode ← 1000 and open file 'stock.txt' in write mode

Else

pcode ← lastproductcode(2) and open file 'stock.txt' in append mode

Step 5: Read product name, category, price and no of stocks

Step 6: Save the detailos in file 'stock.txt'

Step 7: opt1 ← Read value for: Add more(y/n)?

Step 8: If 'y', goto step 5

Step 9: Exit

**[g] Algorithm for UDF displayproduct()**

Step 1: Start

Step 2: Declare file pointer fp

Step 3: Declare pcode, pamt as integers and pprice as float

Step 4: Declare pcat, pname, opt2 as string

Step 5: Open file 'stock.txt' in read mode at file pointer fp

Step 6: If 'stock.txt' has no content

Display "Error"

Goto step 11

Step 7: Read a set of data for each details of one product from file

Step 8: Display the data under respective headings

Step 9: Goto step 7 until end of file

Step 10: Close file fp

Step 11: Get a keypress to exit

**[h] Algorithm for UDF filetostructure()**

Step 1: Start

Step 2: Declare pcode, pamt as integers

Step 3: Declare pcat, pname, opt4 as string

Step 4: Open file 'stock.txt' in read mode at file pointer fp

Step 5: Read details of a product from file 'stock.txt'

Step 6: p[j].pcode ← pcode;

p[j].pamt ← pamt

p[j].pprice ← pprice

p[j].pname ← pname

p[j].pcat ← pcat

j++

Step 7: Close file 'stock.txt'

Step 8: Exit

**[i] Algorithm for UDF lastproductcode(int n)**

Step 1: Start

Step 2: Declare i, j, z, pcode, pamt as integers

Step 3: Declare pcat, pname as string

Step 4: Open file 'stock.txt' in read mode

Step 5: Read for details of a product in file

i++

Step 6: goto step 5 until end of file

Step 7: If n equals 2, goto step 7.1 else goto 10

Step 7.1: Read details of a product from file

Step 7.2: p[j].pcode ← pcode

j++

Step 7.3: Goto step 7.1 until end of file

Step 8: Close file 'stock.txt'

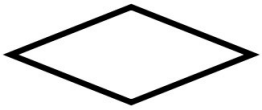
Step 9: Return p[i-1].pcode

Step 10: Return i

Step 11: Exit

## FLOWCHARTS

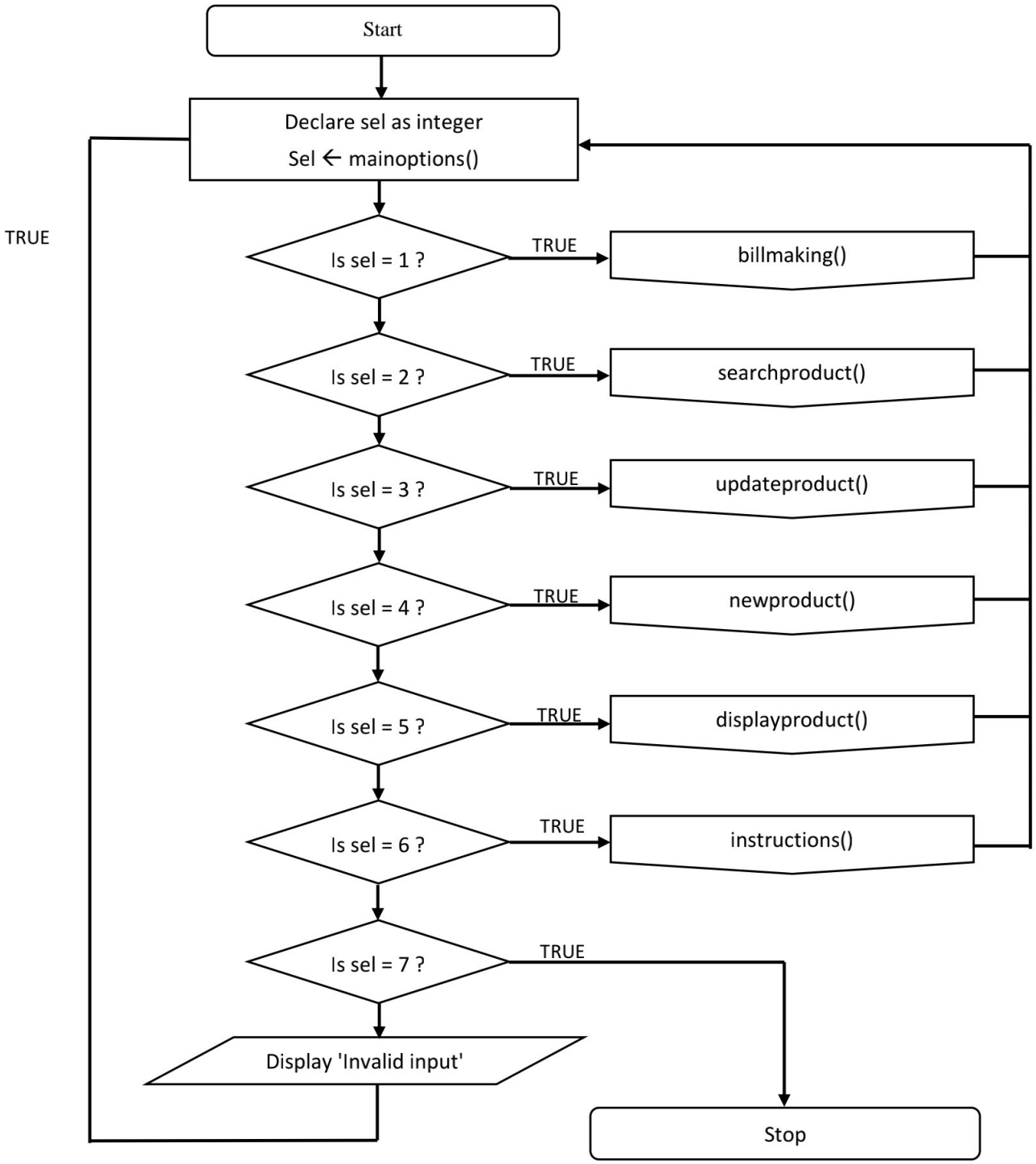
A flowchart is a diagrammatic representation of an algorithm. It illustrates the sequence of operations to be performed to get the solution of problem. Flowcharts facilitates communication between programmers and business people. Flowcharts are usually drawn using standard symbols. Some standard symbols frequently required for flowcharting many computer programs are shown as below:]

| Symbols | Descriptions |
|---|---|
|  | Start / End |
|  | On-page connector |
|  | Decision |
|  | Calculation / operations |
|  | UDF start / end |
|  | Arrow / sequential flow |
|  | Input / Output |

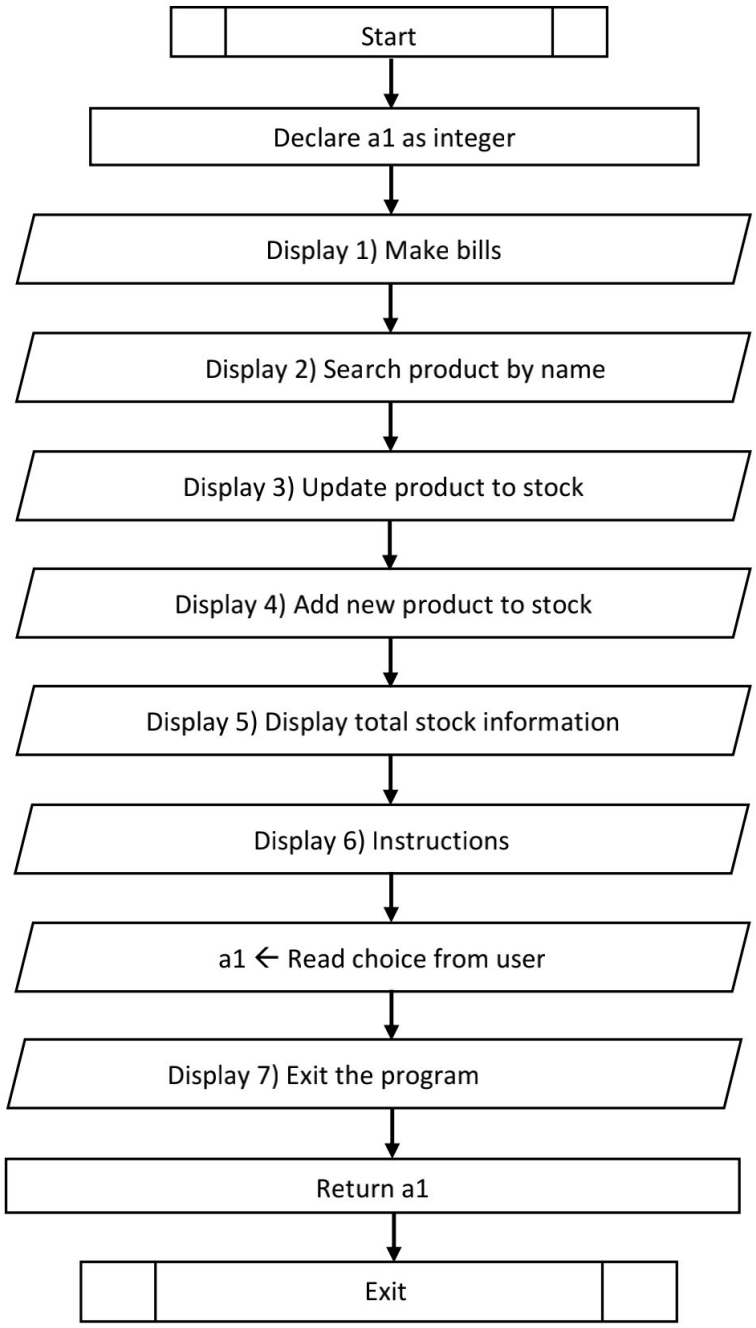
There are many more shapes and their uses but let's not get into details of shapes.

## Some flowcharts of functions used in the programs

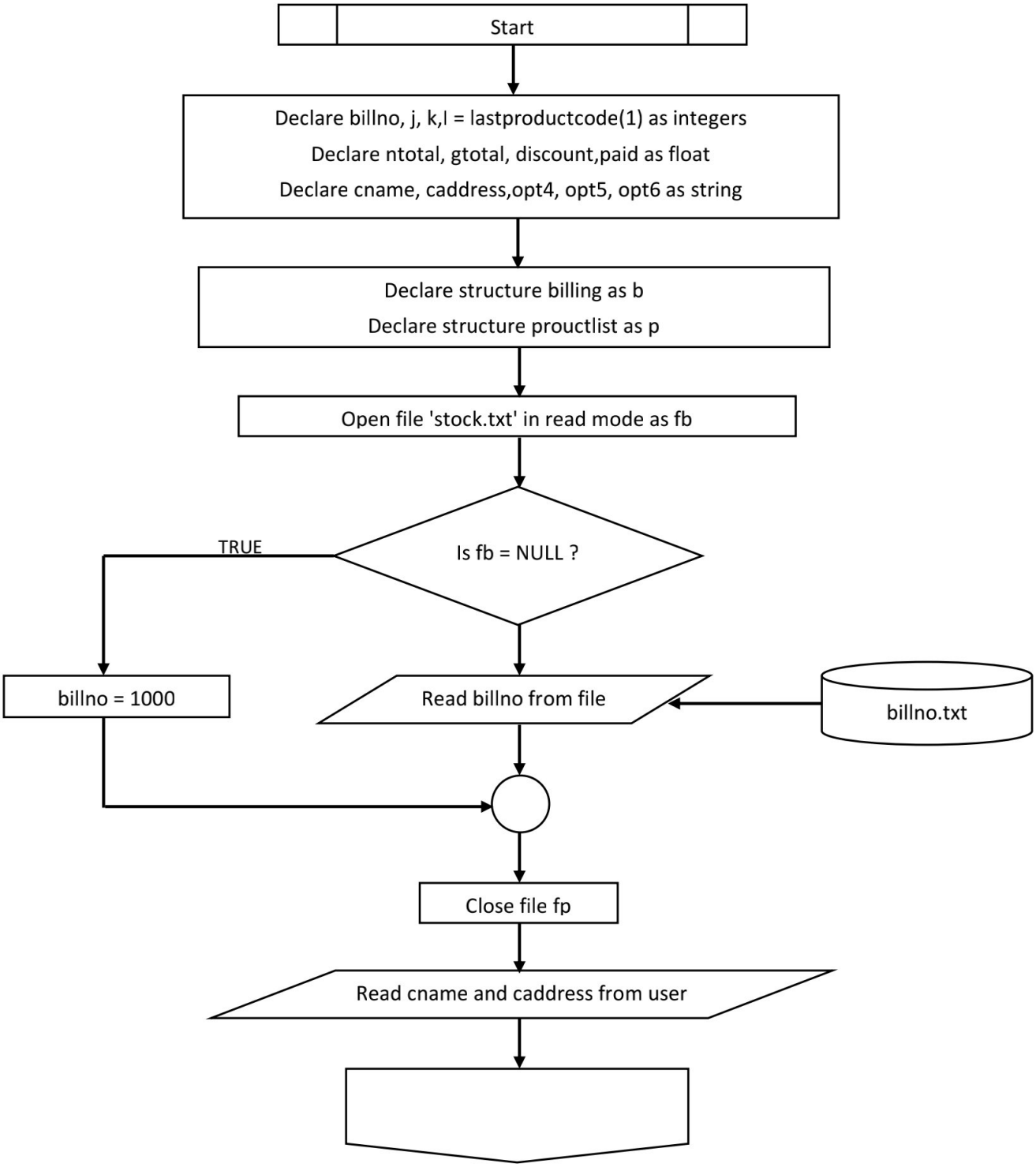
### [a] Main function

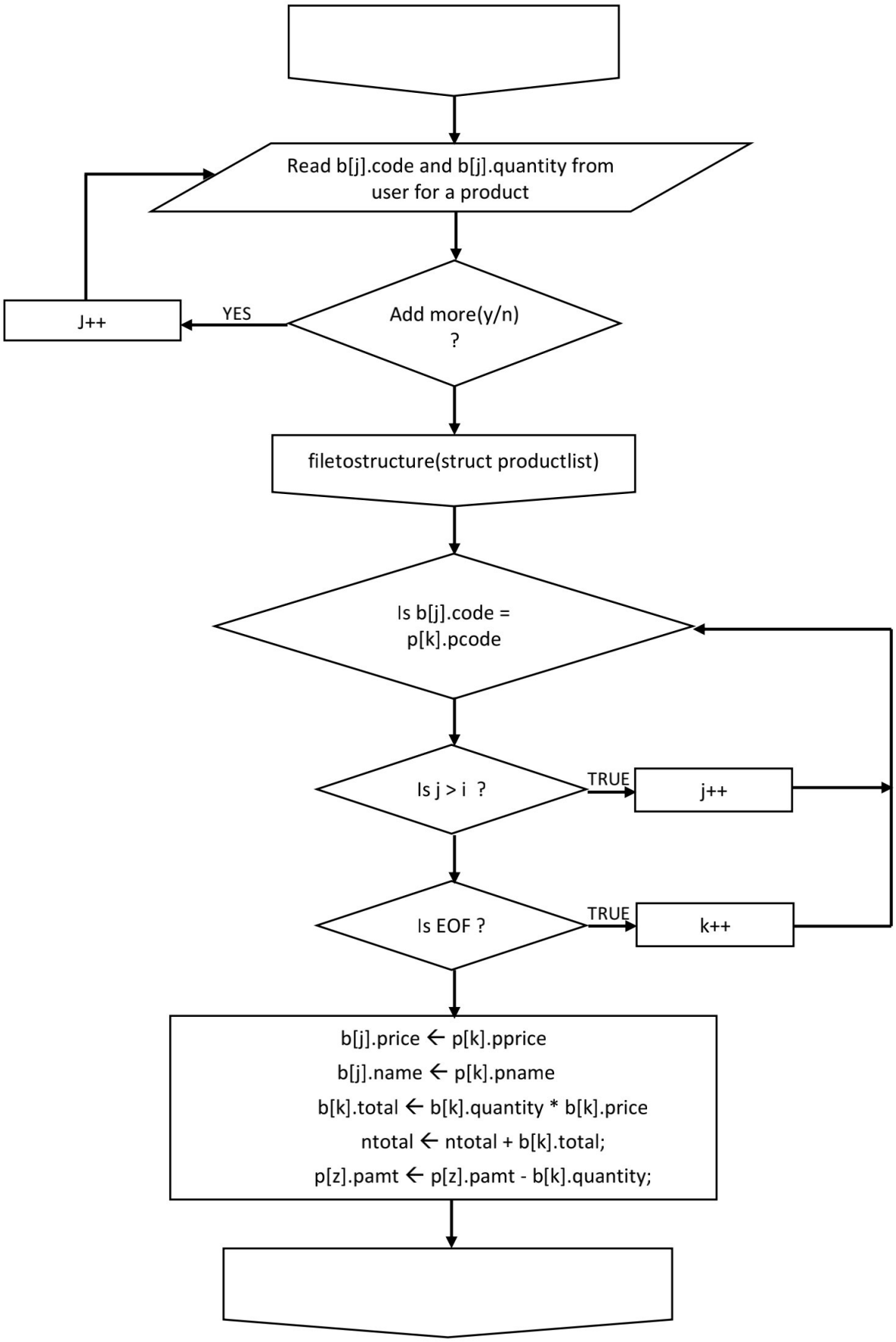


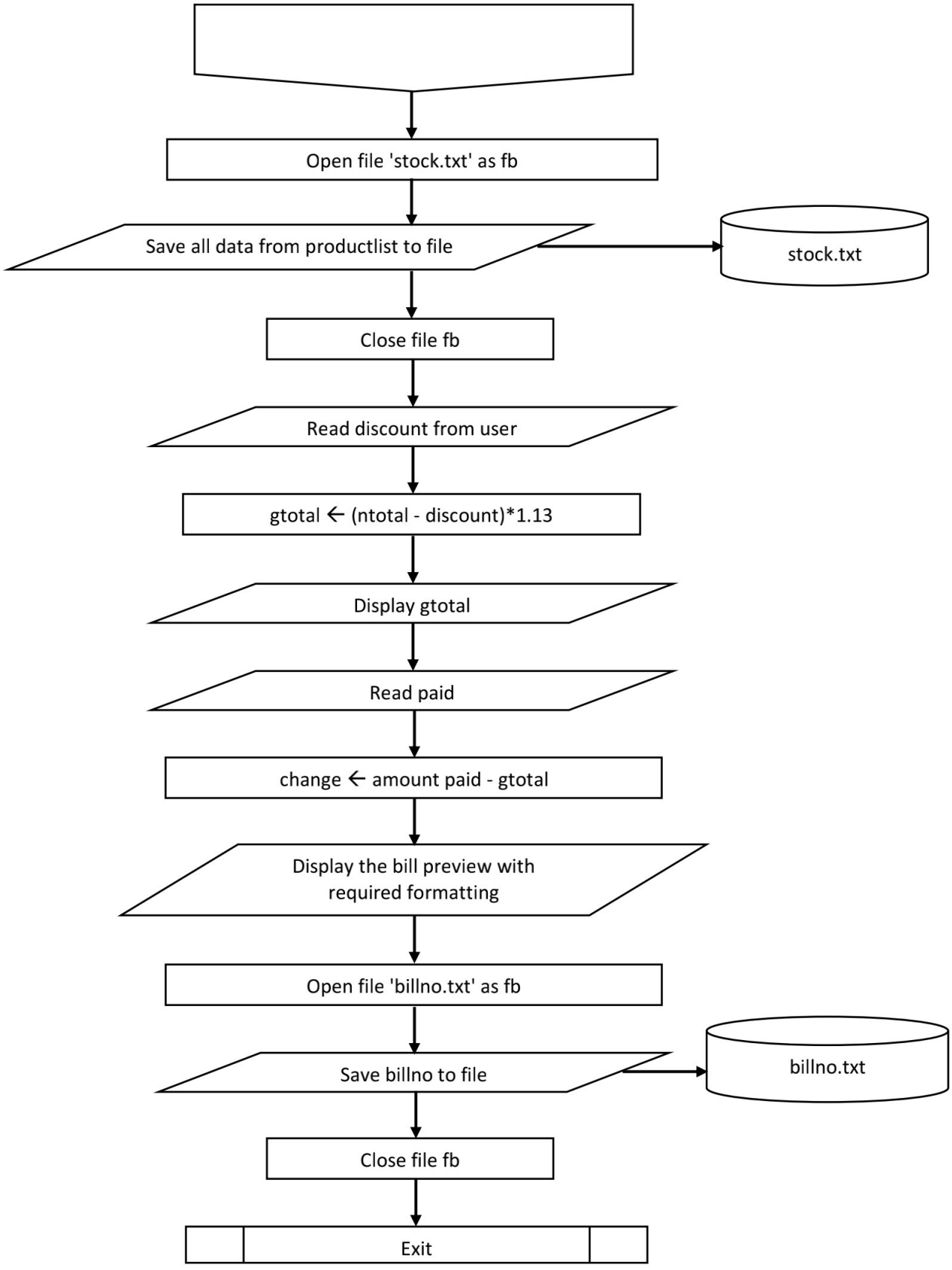
## [b] mainoptions()



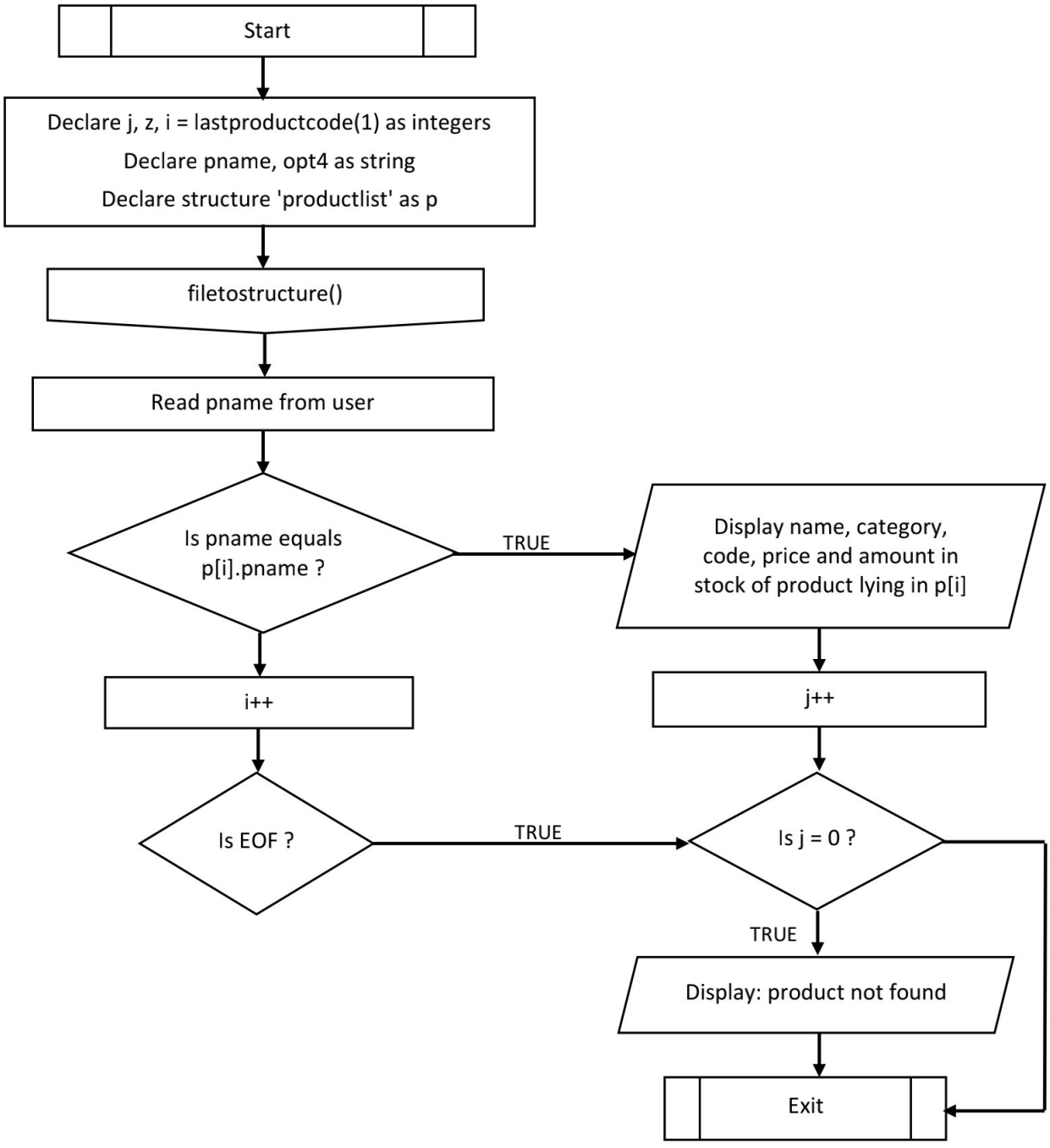
## [c] billmaking()



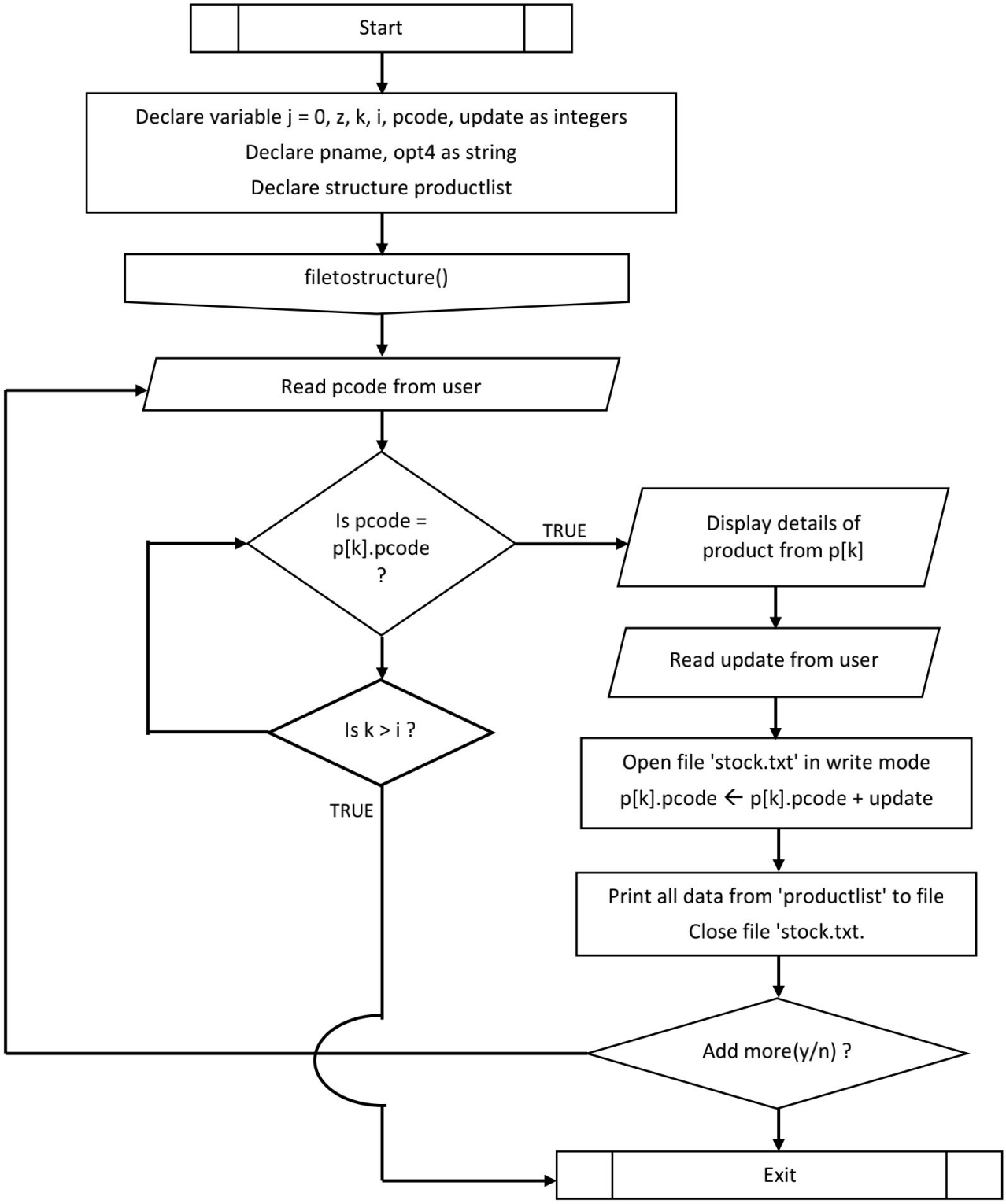




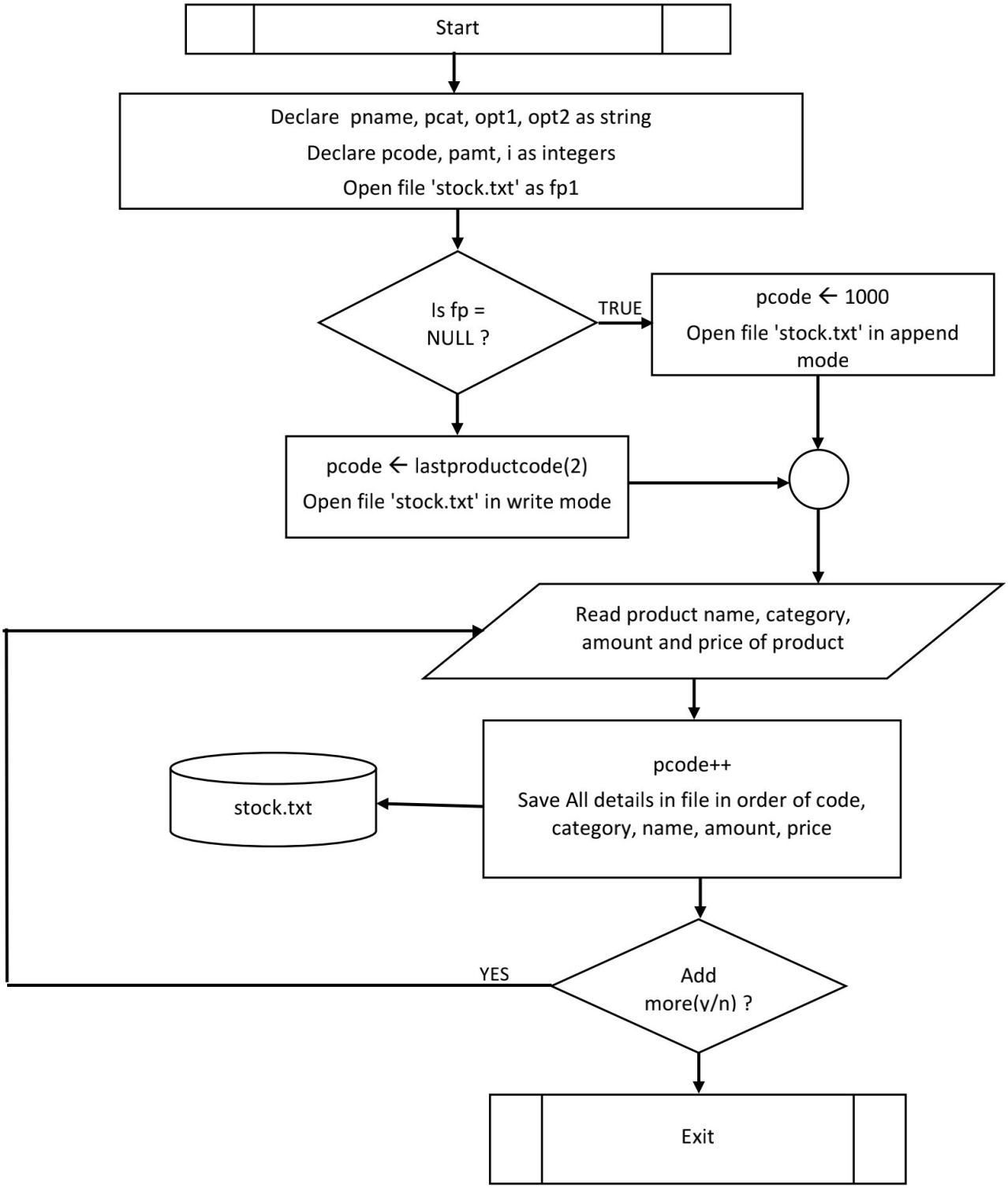
## [d] productsearch()



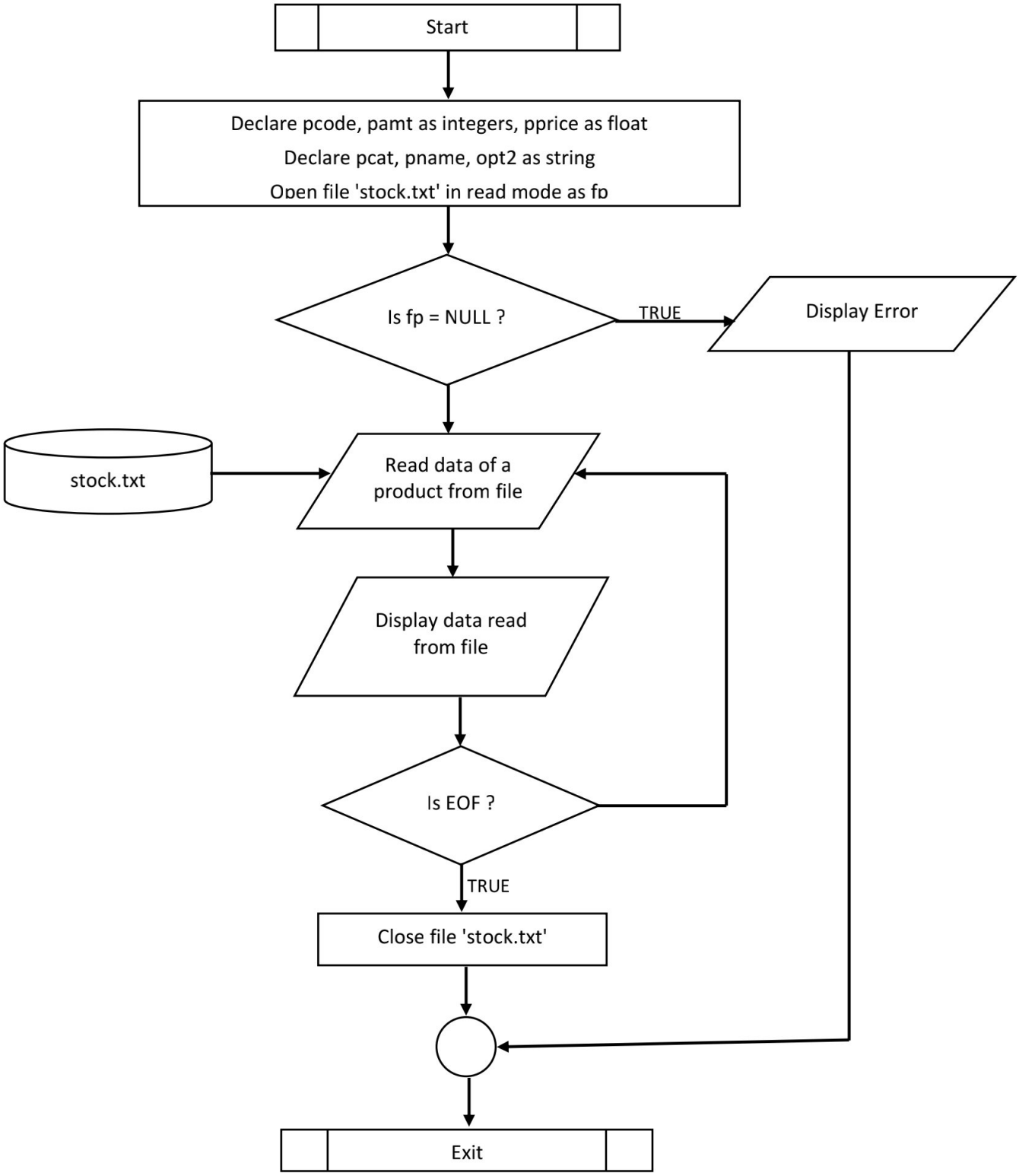
## [e] updateproduct()



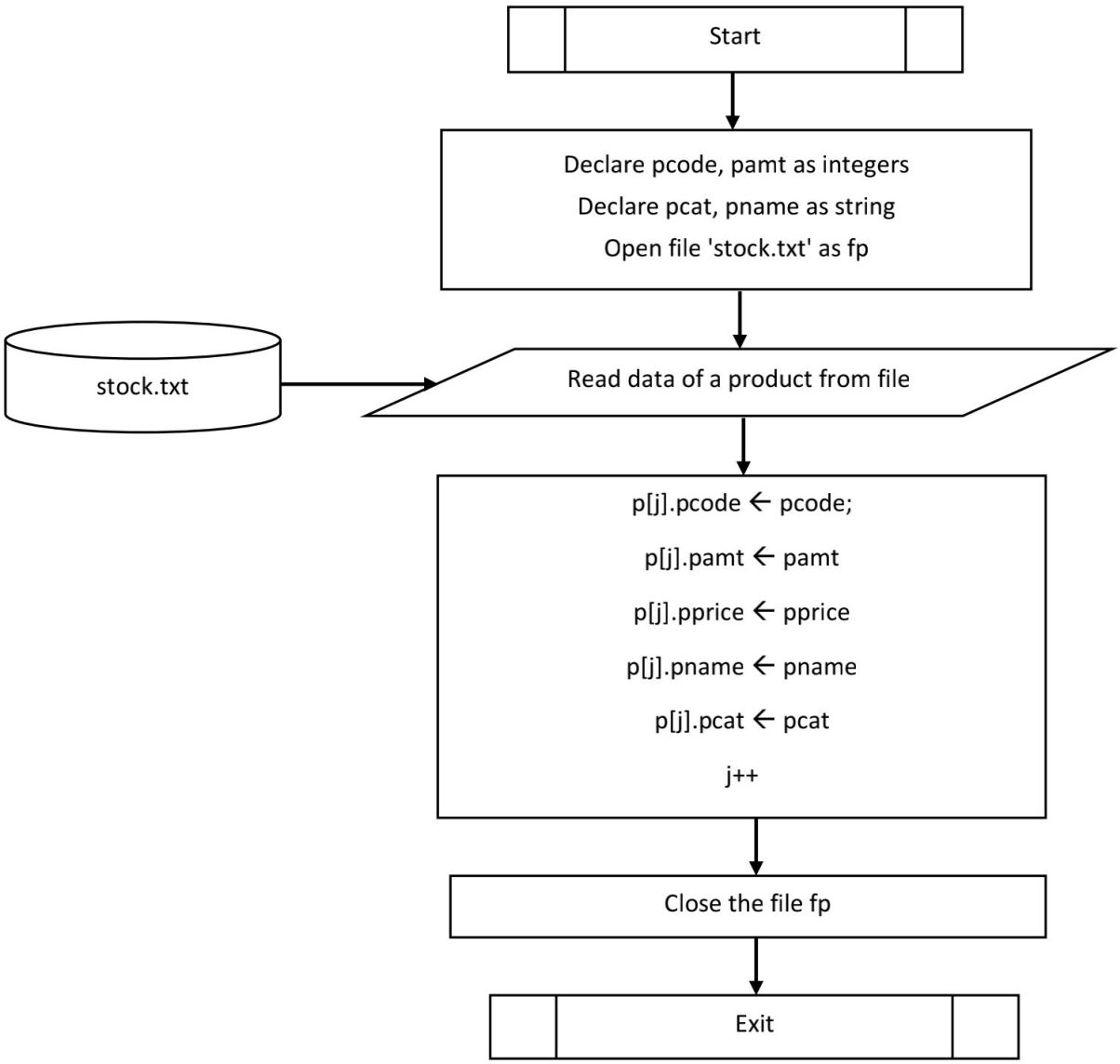
## [f] newproduct()



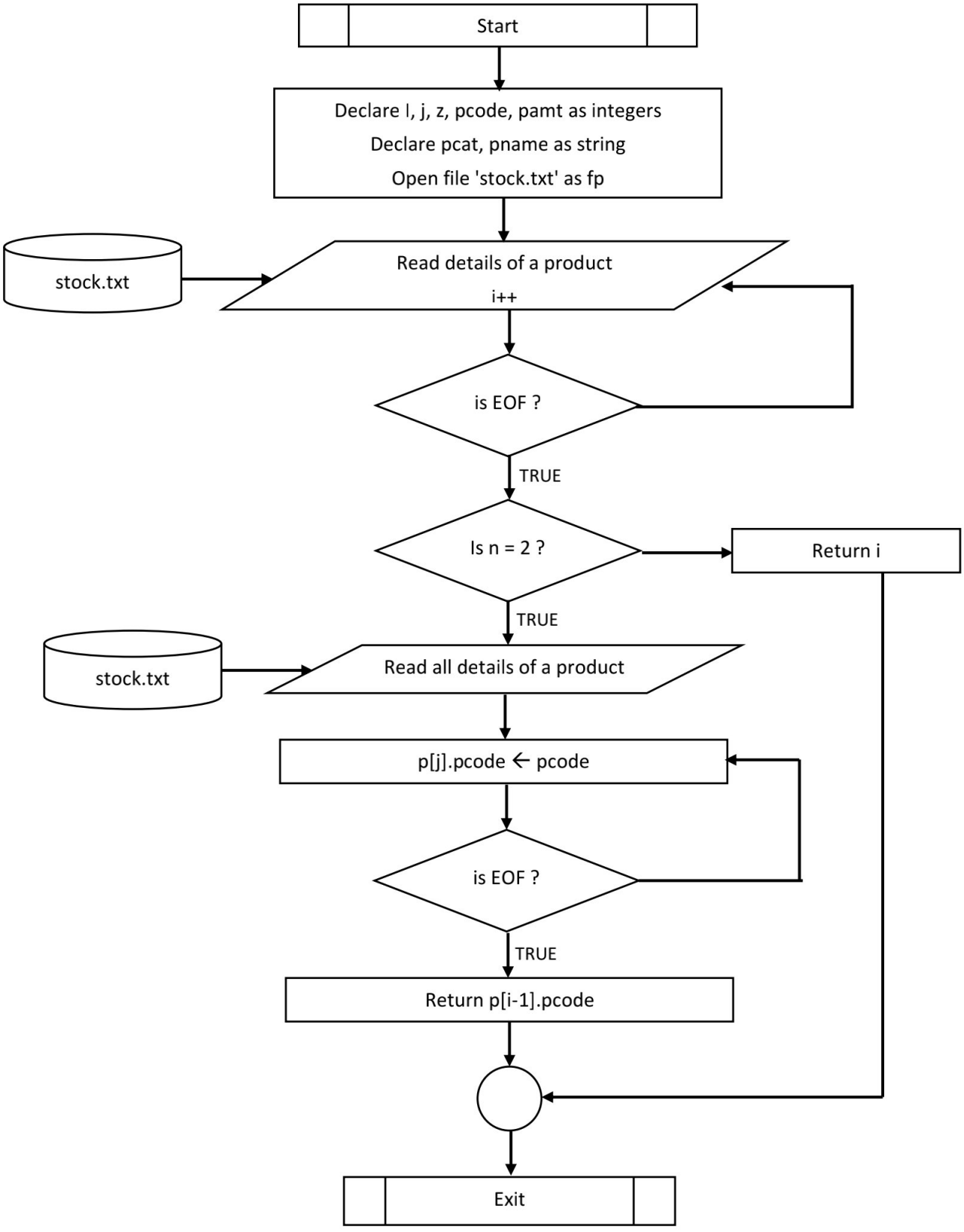
## [g] displayproduct()



## [h] filetostructure(struct productlist)



## [i] lastproductcode(int n)



*Chapter 4*

# IMPLEMENTATION AND CODING

## IMPLEMENTATION

The project has been completed with its objectives as discussed earlier being achieved. The implementation of the plan and the idea of how the final look of the project would be, can be somehow understood from the snapshot of the main screen of the program.

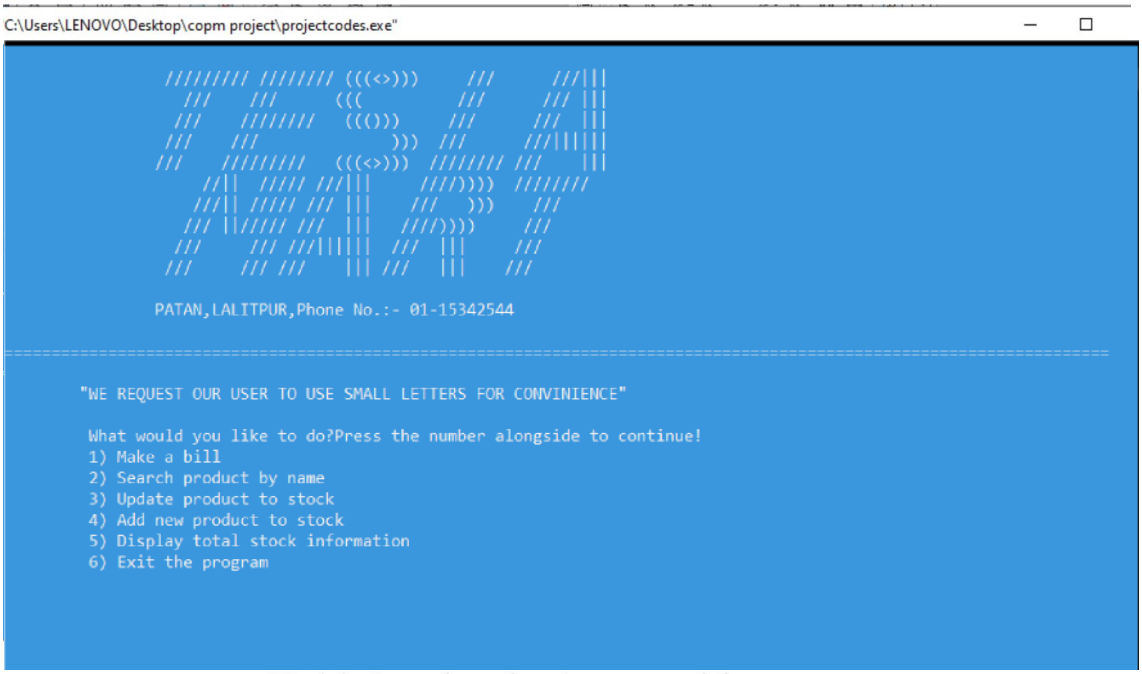



*Fig4.1: Snapshot of main screen of the program*

Here, in the main screen, the program asks user to enter their choice from 1 to 7. The user can choose any option from the menu by entering the corresponding options and the program would proceed to the following entered choice. The user can work in the program as suggested by the program during execution.

Since being a CUI based program, the user has to do all works using their keyboard.

## CODING OF THE PROJECT

The project has been created using C-programming language which is a procedure based programming language. Technically, the program is coded in codeblocks IDM while it is compiled and executed by GNU GCC compiler also known as mingw-32 compiler. The instructions and procedures of the project can be taken from the source codes. Here within this chapter we discuss about the general instructions and User Defined functions, what they do, what inputs does it take and how it works.

### Briefing of the program

- **Header files :** This program has inclusion of stdio.h, string.h, conio.h and stdlib.h header files whose functions has been used throughout the program.
- **Structures :** The program has two different structures defined and used. The first one is 'productlist' that helps to read and write data from the file. The second one is 'billing' that temporarily inputs the data of the products being bought by the customer and stores it for further processing.
- **Main function :** This function has the task for operating the program in the direction as chosen by the user.
- **UDFs :** Various UDFs has been used in the program for various purposes as required by the program. Each UDF has its own set of inputs and does specific task for the program to run efficiently. These had been discussed in detail further below.

### Description of the functions used in the program

**I. int main()**

This is the main function of the program. It sets screen color for the console window and displays the name of supermarket from nameofstore() UDF. And Through the help of mainoption() UDF it allows user to navigate through the program and return back.

**II. int mainoptions()**

This function gives user some options about navigation in the program associated with some number. User can choose from those numbers to do desired works. It returns the entered value back to where it is called from.

**III. void nameofstore()**

This function has the designed name displayed on the console window with some details of the store.

**IV. void billmaking()**

This program takes the input of name and address of the customer. lso, it takes input of the product codes of the product bought by the customer. It compares the product code from the database reads the details of the specific product like price, amount in stock, Name, etc. and does all the calculation for billing like totaling, discount and taxes. It displays the preview of the bill that would be printed.

**V. void productsearch()**

This UDF inputs the name of product being searched from the user and compares it throughout the database that is taken out of the file into structure using UDF filetostructure(). It reads all the details of the product whose name is matched with the input name.

**VI. void productupdate()**

This UDF reads code of the product being updated and no of stocks being added in the store. It compares the code with database and adds the stocks to it and again saves the database back to file with updated data.

**VII. void newproduct()**

This UDF first searches for file 'stock.txt' and if the file is not available or has no data, it takes initial value for product code as 1000. It inputs details from user about each product and saves it to file 'stock.txt' which is open in append mode. It inputs name, category, no of stocks, and price of each new product being added to the database.

**VIII. void displayproducts()**

This function scans details of a product and displays it under respective headings. Again it does same for another file until the end of file.

**IX. void filetostructure()**

This function simply makes the database active to be used in the program. It scans each and every data and stores all the data in variables of structure named 'productlist' which is fed into the function through argument. It scans details of product and stores in variables of structure one by one that can easily be accessed from any place in the program.

**X. void instructions()**

This function just displays the set of instructions for using the program for better use of the program.

**XI. void lastproductcode()**

This function takes an argument of integer type. If input is '1', it counts the number of products in file by continuous scanning of file and returns the number of products in the file 'stock.txt'. And when the input is '2', it scans through all the data and stores the product code of the last product and returns this back to where it is called from.

*Chapter five*

# RESULT AND DISCUSSION

The prepared program is a technical object that needs at least some basic technical idea for operating program. The working process and how the program can be operated is discussed in this chapter along with the result of our actions in the program with their preview that might help someone not related to the program.

## Main Screen

The look of main screen of the program can be learned from snapshot below:

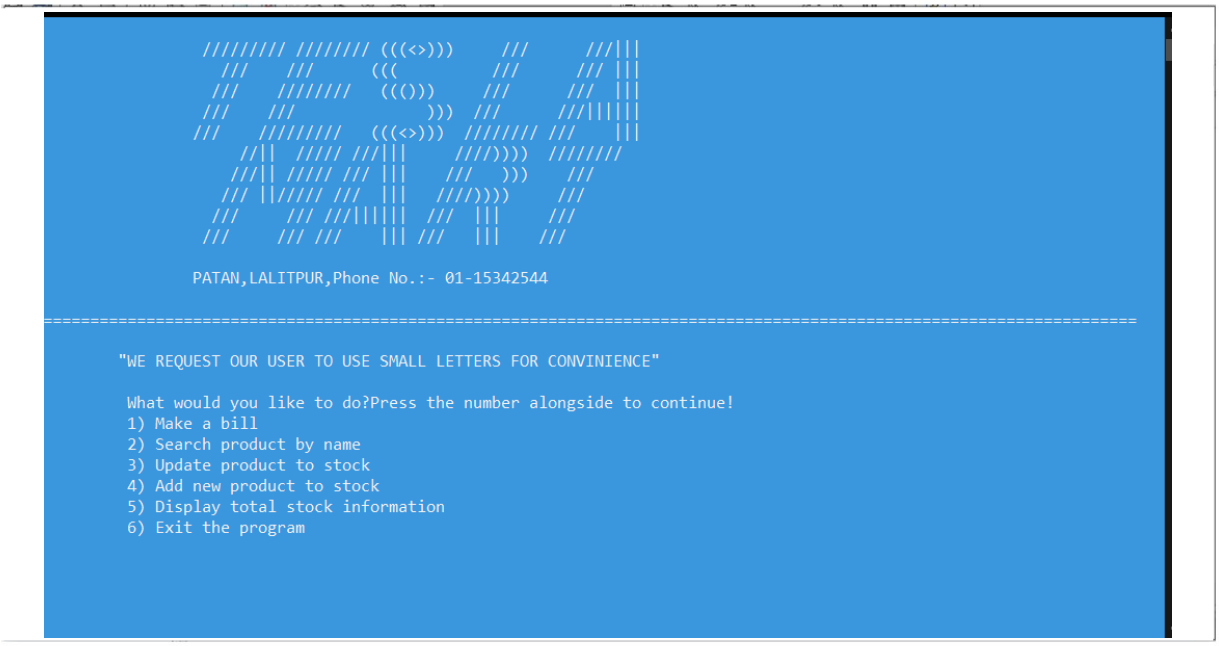



*Fig5.1: Snapshot of main screen of the program*

The program can be continued using the numbers associated with the options. The user can enter any number to proceed to following task.

## Making a Bill

Billing process can be continued from main screen using the number '1'. When the user goes into the billing task, the user needs to enter name and address of customer. Then user needs to enter the respective code of product and their quantity, which can be seen as follows:

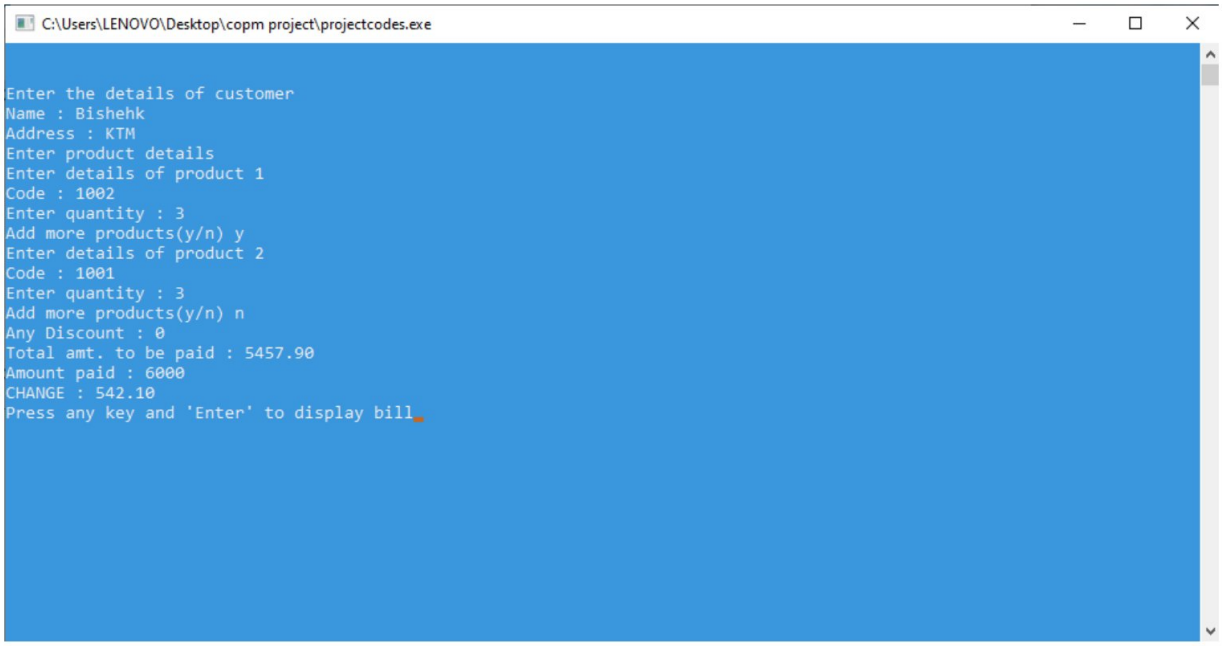



*Fig5.2: Snapshot of data entering screem of bill*

After adding all the products to the list, the program asks the user to enter any discount if given to the customer. After that, the program calculates the amount to be paid by the customer and displays it on screen. Then it asks for the amount of money paid by the customer and calculates the change to be given to customer.

The bill can be previewed after pressing enter which is as follows:

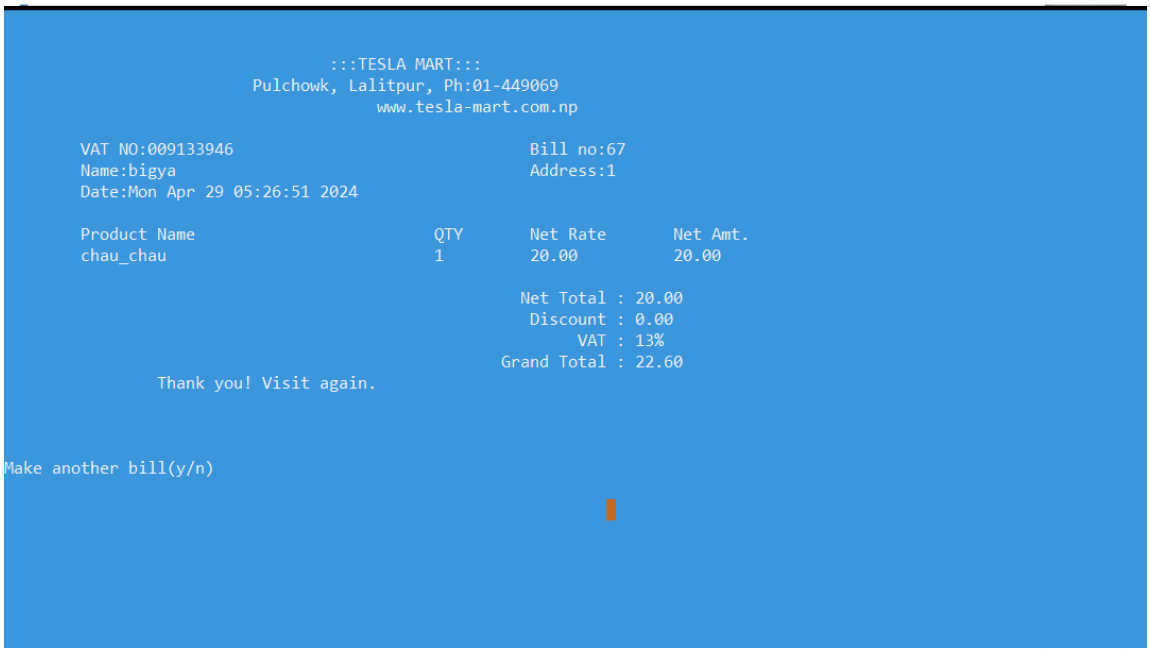



*Fig5.3: Snapshot of preview of bill*

## Searching Product

Sometimes we need to look for some items and its details in the store to know better about it. The operation of search product does the same for us. The user just needs to enter the name of product being searched and the program will scan for it in the database and give the details if it is present in the database. The preview of the screen after search can be understood from picture below:

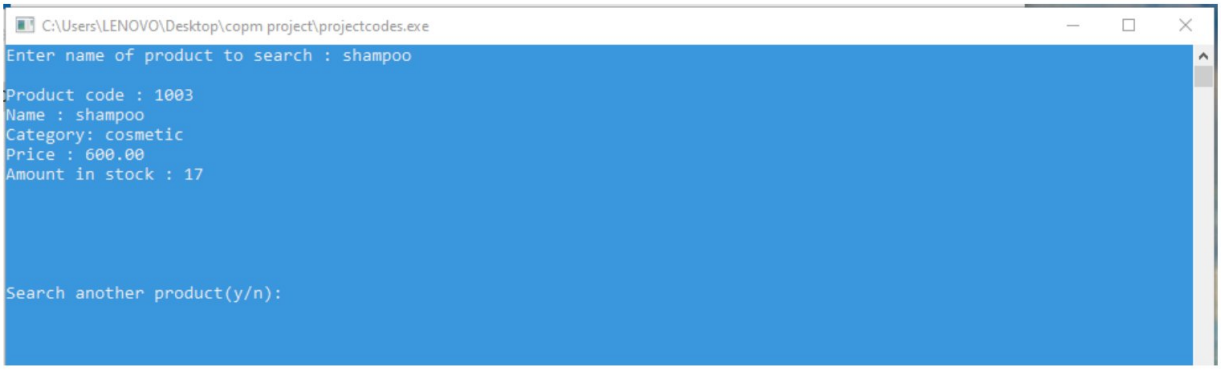



*Fig5.4: Snapshot of preview of searching product*

## Updating product

No store in the world doesn't run out of resource to sell. They also need to add more products to make more profit. For this they need to regularly add products in the store to continue demand and supply cycle. For this purpose the program has the function of update product that can be used by entering the option '3' in the main screen. The user needs to first enter the code of product being updated in the stock and then the program will display its details to confirm the product and then asks for the no of products being added and adds to the stock.

The preview of the update screen can be seen below:

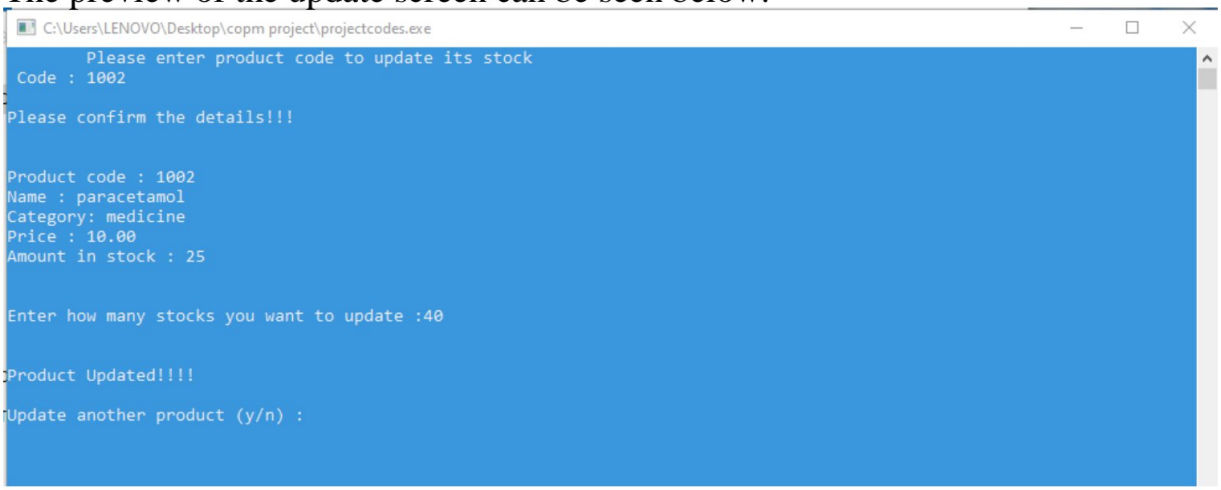



*Fig5.5: Snapshot of preview update screen*

## New product

For increasing the productivity and popularity of the supermarket, it must always sell all products. The newly launched products need to be added to the store so that it can fulfill all the needs of the customer. For this purpose, new product function is present in the program.

After the user chooses for the option '4', the program proceeds to the new product function. The program asks the user for each details of the new product that includes name, category, price and amount of products and then it saves the product at the end of file where all the products are stored. The preview of adding new product can be seen below:

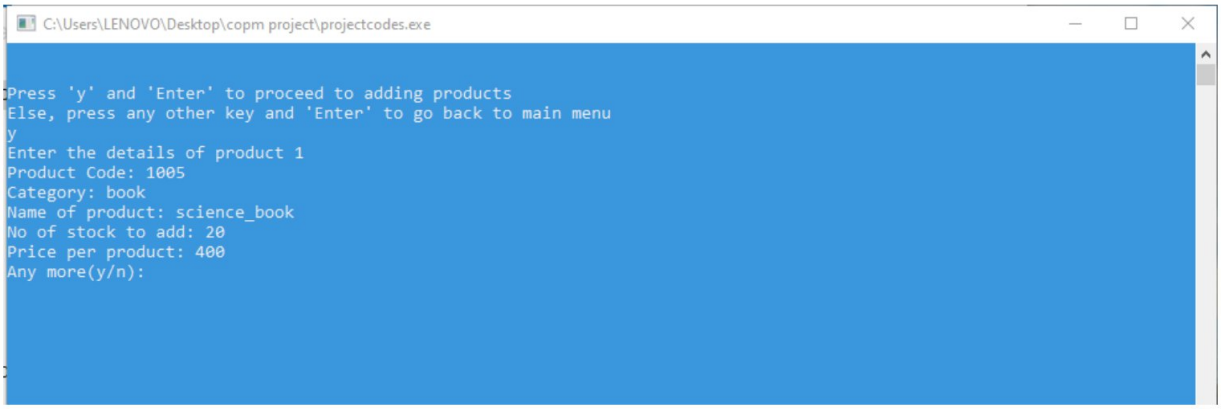



*Fig5.3: Snapshot of preview adding new product*

## Display product

In order to know about all the products in the store, we need to have a detailed preview of the database. The display product function does the same for the user. The user needs to enter he number '5' in the main screen to proceed to this function. The preview off displaying the database can be known from here:

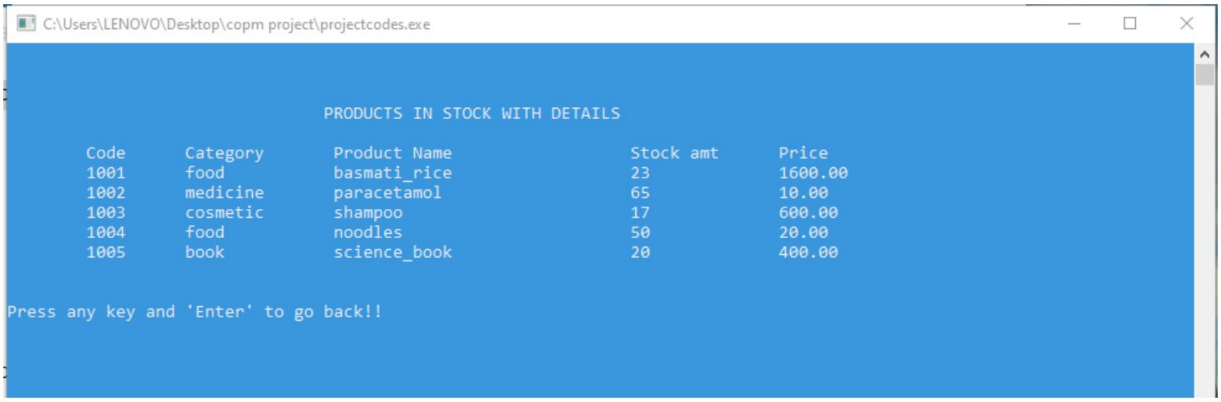



*Fig5.3: Snapshot of preview display od database*

## Instructions

On entering the number '6', the program executes the instructions function. This portion of the program just displays some rules that must be followed during using of the program. It's preview can be done below:

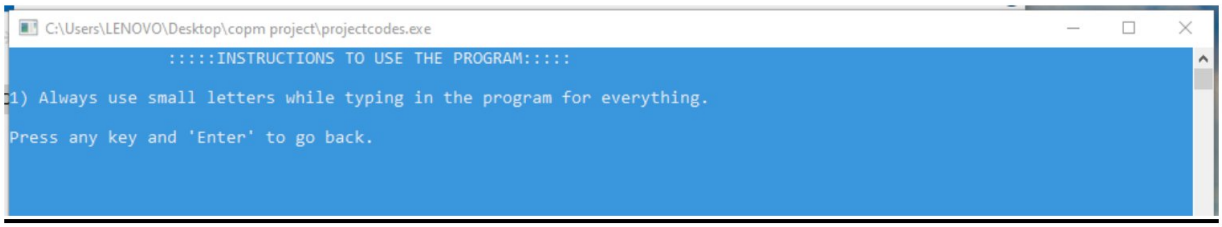



*Fig5.3: Snapshot of preview of instructions*

## Exit the program

On choosing the option '7' from the main screen, the program stops and exits the console window.

## CONCLUSION

After proper testing and executing the program, following conclusions can be made:

- This program is capable of making bills with internal calculations.
- This program can search for any product whose information is needed to be found.
- This program also adds stocks to the existing program.
- This program can add new products and their details in the file.
- Users can see all the products in the database at one place.
- This program also helps in understanding the use of various functions in C programming.

# REFERENCES


**Books**

- Kantekar, Y (2014), *Let Us C,* Infinity Science Press
- Kurniawan, A, *Learning C by Example*

**Website**

- www.programiz.com
- www.trello.com
- www.lucidchart.com

# Appendix

Here's the code to our program-

```
#include <stdio.h>
#include <time.h>
#include <conio.h>
#include <string.h>
#include <stdlib.h>
struct billing
{
   int code;
   int quantity;
   char name[25];
   float price;
   float total;
};
struct productslist
{
   int pcode;
   char pcat[10];
   char pname[25];
   int pamt;
   float pprice;
};
void Nameofstore();
int mainoptions();
void billmaking();
void productsearch();
void productupdate();
void newproduct();
void displayproducts();
```

```
void filetosructure(struct productslist[]);
int lastproductcode(int);
int main()
{
label1:
   system("color 3F");
   Nameofstore();
   int sel, opt2;
   sel = mainoptions();
   if (sel == 1)
   {
      billmaking();
      system("cls");
      goto label1;
   }
   else if (sel == 2)
   {
      productsearch();
      system("cls");
      goto label1;
   }
   else if (sel == 3)
   {
      productupdate();
      system("cls");
      goto label1;
   }
   else if (sel == 4)
   {
      newproduct();
      system("cls");
```

```
        goto label1;
    }
    else if (sel == 5)
    {
        displayproducts();
        system("cls");
        goto label1;
    }

    else if (sel == 6)
    {
        exit(0);
    }
    else
    {
        system("cls");
        printf("Invalid Input!!!Please enter integers from 1-6\n\n\n\n\n");
        goto label1;
    }
}
int mainoptions()
{
    int a1;
    printf("\t\"WE REQUEST OUR USER TO USE SMALL LETTERS FOR CONVINIENCE\"\n\n");
    printf("\t What would you like to do?Press the number alongside to continue!\n");
    printf("\t 1) Make a bill\n");
    printf("\t 2) Search product by name\n");
    printf("\t 3) Update product to stock\n");
    printf("\t 4) Add new product to stock\n");
    printf("\t 5) Display total stock information\n");
    printf("\t 6) Exit the program\n\n");
```

```
    scanf("%d", &a1);
    return a1;
}
void Nameofstore()
{
    printf("\n");
    printf("\t\t //////// /////// (((<>)))    ///     ///|||\n");
    printf("\t\t  ///   ///     (((        ///     /// |||\n");
    printf("\t\t  ///   ////////  ((()))    ///     ///  |||\n");
    printf("\t\t ///   ///           ))) ///     ///||||||\n");
    printf("\t\t///   ////////  (((<>))) /////// ///    |||\n");

    printf("\t\t    //|| ///// ///|||    ////))))  /////// \n");
    printf("\t\t   ///|| ///// /// |||   ///   )))   /// \n");
    printf("\t\t  /// ||////// ///  |||  ////))))    /// \n");
    printf("\t\t ///    /// ///||||||  ///  |||    /// \n");
    printf("\t\t ///    /// ///   ||| ///   |||   /// \n\n");
    printf("\t\tPATAN,LALITPUR,Phone No.:- 01-15342544\n\n");

printf("================================================================================
=========================================\n\n");
}
void billmaking()
{
label4:
    system("cls");
    time_t t;
    time(&t);
    int m = 0, billno;
    FILE *fb;
    fb = fopen("billno.txt", "r");
```

```
if (fb == NULL)
{
   billno = 10000;
}
else
{
   fscanf(fb, "%d", &billno);
}
fclose(fb);
char cname[25], caddress[25], opt4[3], opt5[3], opt6[3];
printf("\n\nEnter the details of customer\n");
printf("Name : ");
scanf("%s", cname);
printf("Address : ");
scanf("%s", caddress);

struct billing b[100];
printf("Enter product details\n");
do
{
   printf("Enter details of product %d\nCode : ", m + 1);
   scanf("%d", &b[m].code);
   printf("Enter quantity : ");
   scanf("%d", &b[m].quantity);
   m++;
   printf("Add more products(y/n) ");
   scanf("%s", &opt5);
} while (strcmp(opt5, "n"));
int j = 0, z, k, i = lastproductcode(1), pcode;
float ntotal = 0, gtotal = 0, discount, paid;
struct productslist p[i];
```

```
filetostructure(p);
for (z = 0; z < i; z++)
{
   for (k = 0; k < m; k++)
   {
      if (p[z].pcode == b[k].code)
      {
         b[k].price = p[z].pprice;
         strcpy(b[k].name, p[z].pname);
         b[k].total = b[k].quantity * b[k].price;
         ntotal = ntotal + b[k].total;
         p[z].pamt = p[z].pamt - b[k].quantity;
      }
   }
}
FILE *fp;
fp = fopen("resource.txt", "w");
for (j = 0; j < i; j++)
{
   fprintf(fp, " %d %s %s %d %f", p[j].pcode, p[j].pcat, p[j].pname, p[j].pamt, p[j].pprice);
}
fclose(fp);
printf("Any Discount : ");
scanf("%f", &discount);
gtotal = (ntotal - discount) * 1.13;
printf("Total amt. to be paid : %.2f\n", gtotal);
printf("Amount paid : ");
scanf("%f", &paid);
printf("CHANGE : %.2f\n", paid - gtotal);
printf("Press any key and 'Enter' to display bill");
scanf("%s", &opt6);
```

```
    system("cls");
    printf("\n\n");
    printf("%34s:::TESLA MART:::\n", "");
    printf("%26sPulchowk, Lalitpur, Ph:01-449069\n", "");
    printf("%39swww.tesla-mart.com.np\n\n", "");
    printf("\tVAT NO:009133946 %-30sBill no:%d\n", "", ++billno);
    printf("\tName:%-42sAddress:%-18.15s\n", cname, caddress);
    printf("\tDate:%s\n", ctime(&t));
    printf("\tProduct Name%-25sQTY%-7sNet Rate%-7sNet Amt.\n", "", "", "");
    for (int r = 0; r < m; r++)
    {
        printf("\t%-37.25s%-10d%-15.2f%.2f\n", b[r].name, b[r].quantity, b[r].price, b[r].total);
    }
    printf("\n\t%55s : %.2f", "Net Total", ntotal);
    printf("\n\t%55s : %.2f", "Discount", discount);
    printf("\n\t%55s : %s", "VAT", "13%");
    printf("\n\t%55s : %.2f", "Grand Total", gtotal);
    printf("\n\t\tThank you! Visit again.");

    fb = fopen("billno.txt", "w");
    fprintf(fb, "%d", billno);
    fclose(fb);
    printf("\n\n\n\nMake another bill(y/n)");
    scanf("%s", &opt4);
    if (strcmp(opt4, "y") == 0)
    {
        goto label4;
    }
}
void productsearch()
{
```

```
    int j = 0, z, i = lastproductcode(1), pcode;
    char pname[25], opt4[3];
    struct productslist p[i];
    filetostructure(p);
label4:
    system("cls");
    printf("Enter name of product to search : ");
    scanf("%s", &pname);
    for (z = 0; z < i; z++)
    {
        if (strcmp(p[z].pname, pname) == 0)
        {
            printf("\nProduct code : %d\n", p[z].pcode);
            printf("Name : %s\n", p[z].pname);
            printf("Category: %s\n", p[z].pcat);
            printf("Price : %.2f\n", p[z].pprice);
            printf("Amount in stock : %d\n\n\n", p[z].pamt);
            j++;
        }
    }
    if (j == 0)
    {
        printf("\n\nProduct not found!!!");
    }
    printf("\n\n\nSearch another product(y/n):");
    scanf("%s", &opt4);
    if (strcmp(opt4, "y") == 0)
    {
        j = 0;
        goto label4;
    }
```

```
}
void productupdate()
{
   int j = 0, z, k, i = lastproductcode(1), pcode, update;
   char pname[25], opt4[3];
   struct productslist p[i];
   filetostructure(p);
   system("cls");
label4:
   printf("\tPlease enter product code to update its stock\n ");
   printf("Code : ");
   scanf("%d", &pcode);
   for (z = 0; z < i; z++)
   {
      if (p[z].pcode == pcode)
      {
         printf("\nPlease confirm the details!!!\n\n");
         printf("\nProduct code : %d\n", p[z].pcode);
         printf("Name : %s\n", p[z].pname);
         printf("Category: %s\n", p[z].pcat);
         printf("Price : %.2f\n", p[z].pprice);
         printf("Amount in stock : %d\n\n\n", p[z].pamt);
         j++;
         k = z;
      }
   }
   if (j == 0)
   {
      printf("\n\nProduct not found!!!");
   }
   else
```

```
    {
        printf("Enter how many stocks you want to update :");
        scanf("%d", &update);
        p[k].pamt = p[k].pamt + update;
        printf("\n\nProduct Updated!!!!");
    }
    FILE *fp;
    fp = fopen("resource.txt", "w");
    for (j = 0; j < i; j++)
    {
        fprintf(fp, " %d %s %s %d %f", p[j].pcode, p[j].pcat, p[j].pname, p[j].pamt, p[j].pprice);
    }
    fclose(fp);
    printf("\n\nUpdate another product (y/n) : ");
    scanf("%s", &opt4);
    if (strcmp(opt4, "y") == 0)
    {
        j = 0;
        goto label4;
    }
}
void newproduct()
{
    int pcode, pamt, i = 1;
    float pprice;
    char pname[25], pcat[10], opt1, opt2[5];
    system("cls");
    printf("\n\nPress 'y' and 'Enter' to proceed to adding products\n");
    printf("Else, press any other key and 'Enter' to go back to main menu\n");
    scanf("%s", &opt2);
    if (strcmp(opt2, "y") == 0)
```

```
{
   pcode = lastproductcode(2);
   FILE *fp;
   FILE *fp1;
   fp1 = fopen("resource.txt", "r");
   if (fp1 == NULL)
   {
      pcode = 1000;
      fp = fopen("resource.txt", "w");
   }
   else
   {
      fp = fopen("resource.txt", "a");
   }
   fclose(fp1);
   do
   {
      pcode++;
      printf("Enter the details of product %d\n", i++);
      printf("Product Code: %d\n", pcode);
      printf("Category: ");
      scanf("%s", &pcat);
      printf("Name of product: ");
      scanf("%s", &pname);
      printf("No of stock to add: ");
      scanf("%d", &pamt);
      printf("Price per product: ");
      scanf("%f", &pprice);
      fprintf(fp, " %d %s %s %d %f", pcode, pcat, pname, pamt, pprice);
      printf("Any more(y/n):");
      scanf("%s", &opt1);
```

```
        } while (opt1 != 'n');
        printf("\n\nProducts added!!!\n");
        fclose(fp);
        getchar();
    }
}
void displayproducts()
{
    FILE *fp;
    int pcode, pamt;
    char pcat[10], pname[25], opt2[5];
    float pprice;
    system("cls");
    fp = fopen("resource.txt", "r");
    if (fp == NULL)
    {
        printf("Error opening file.....");
        goto label2;
    }
    printf("\n\n\n\t\t\t\tPRODUCTS IN STOCK WITH DETAILS\n\n");
    printf("\t%-10s%-15s%-30s%-15s%s\n", "Code", "Category", "Product Name", "Stock amt",
"Price");
    while (!feof(fp))
    {
        fscanf(fp, "%d%s%s%d%f", &pcode, &pcat, &pname, &pamt, &pprice);
        printf("\t%-10d%-15s%-30s%-15d%.2f\n", pcode, pcat, pname, pamt, pprice);
    }
label2:
    fclose(fp);
    printf("\n\nPress any key and 'Enter' to go back!!");
    scanf("%s", &opt2);
```

```
}
int lastproductcode(n)
{
   FILE *fp;
   int i = 0, j = 0, z;
   int pcode, pamt;
   char pcat[10], pname[25];
   float pprice;
   fp = fopen("resource.txt", "r");
   if (fp == NULL)
   {
      z = 0;
      goto label3;
   }
   while (!feof(fp))
   {
      fscanf(fp, "%d%s%s%d%f", &pcode, &pcat, &pname, &pamt, &pprice);
      i++;
   }
   fclose(fp);
   if (n == 2)
   {
      fp = fopen("resource.txt", "r");
      if (fp == NULL)
      {
         z = 0;
         goto label3;
      }
      struct productslist p[i + 1];
      while (!feof(fp))
      {
```

```
            fscanf(fp, "%d%s%s%d%f", &pcode, &pcat, &pname, &pamt, &pprice);
            p[j].pcode = pcode;
            j++;
        }
        fclose(fp);
        return p[i - 1].pcode;
    }
    return i;
label3:
    if (z == 0)
    {
        return 000000;
    }
}

void filetostructure(struct productslist p[])
{
    int pcode, pamt, j = 0;
    char pcat[10], pname[25], opt4[3];
    float pprice;
    FILE *fp;
    fp = fopen("resource.txt", "r");
    while (!feof(fp))
    {
        fscanf(fp, "%d%s%s%d%f", &pcode, &pcat, &pname, &pamt, &pprice);
        p[j].pcode = pcode;
        p[j].pamt = pamt;
        p[j].pprice = pprice;
        strcpy(p[j].pname, pname);
        strcpy(p[j].pcat, pcat);
        j++;
```

```
    }
    fclose(fp);
}
```