Compare the performances of greedy, backtracking, dynamic programming, and brute-force approach to solve 0/1 knapsack problem.

RAMACHANDRAN R

Abstract:

A knapsack is a sack or bag that can be filled up to the given maximum weight. Here, each of the object have some value (i.e. worth) and some weight. Also, these objects are indivisible. So, we can't divide the object into its fractional parts accordingly to fill the sack. This is the reason why we give the name 0/1 knapsack, which means '0' indicates that the particular object is not taken and '1' indicates that the particular object is taken in the sack (like we use in digital logics). Our aim is to fill the sack such that we get the maximum profit(value) and the weight of all the objects that are taken should be less than are equal to the maximum capacity of the bag (i.e. maximum weight capacity). In mathematical terms,

$$\text{Maximum}(\Sigma p_i x_i) \text{ and } \Sigma w_i x_i <= W$$

Where,

pi = The profit of that particular object.

wi = The weight of that particular object.

xi = 1 or 0(i.e., object is taken or left behind respectively)

W = Maximum weight capacity of the sack.

For example, let us consider a situation where we need to pack a bag such that the contents provide maximum value without exceeding our bag's capacity. We assume that we have a Knapsack that can hold at most 5 Kgs and have the following items.

THINGS	VALUE	WEIGHT
Clock	450	0.5 Kg
Painting	500	0.5 Kg
Radio	1250	1 Kg
Laptop	45000	2 Kgs
Grinder	5000	2 Kgs
DVD Player	5000	1 Kg

To solve this problem, we use different kinds of algorithms like backtracking, greedy, dynamic programming, brute-force etc.

The main goal is to present the comparative study of the approaches to find the performance of the different algorithms used to solve the 0/1 Knapsack problem, based on the time complexity of each algorithm.

AIM:

To compare the performances of greedy, backtracking, dynamic programming, and brute-force approach to solve 0/1 knapsack problem.

APPLICABILITY:

The knapsack problem can be widely applied in:

1. Resource Allocation
2. Job Scheduling
3. Capital Budgeting
4. Investment Decisions
5. Project Selection

The proposed application is using knapsack in a startup company.

The initial investment is taken to be the maximum weight and the stock items have their own individual price(weight) and their own profit(value).

So being a member of the start up company, we need to utilize the initial investment in the best way possible by buying those stock items such that we get maximum profit.

Here we use the 0/1 knapsack problem and solve the issue such that the company benefits maximum from it.

In today’s world where everyone is on a look out to start something new, something of their own, it is essential to have proper planning on what to invest on such that they benefit maximum from it. People invest a good amount but some end up with losses since they were unable to decide on the right stocks to invest on.

Since there is a large database available, it is also required to use an algorithm that works fast and efficiently. Hence, we compare the working of the four algorithms and find out the best algorithm in order to solve the above application of knapsack.

INTRODUCTION:

In our program, we take-

Maximum weight: Initial Investment

Individual weights: Individual prices of the stock items

Individual values: Individual profits on selling the stock items.

So, our aim is to get maximum profit by using our investment wisely.

The program uses the concept of data structures as well as algorithms.

To solve the knapsack, we use the concept of queues and linked list data structures and do dynamic programming in order to get maximum profit for the company.

The comparison of the algorithms is the concept derived from algorithms where we compare the time taken to compile (time complexity) when the data is passed using brute force, dynamic programming, greedy and back tracking approaches.

While we do this, we find out that **dynamic programming is the fastest** and the most efficient way of solving the knapsack problem due to the least time taken to compile.

We have various member functions in order to add stock items, display stock items, delete stock items, alter the price or the profit of a stock item and so on, adding functionality to our program.

EXISTING METHODS AND THEIR DRAWBACKS:

1. THE BRUTE FORCE APPROACH

Brute force is a straightforward approach to solving a problem, usually directly based on the problem's statement and definitions of the concepts involved.

Algorithm:

- If there are n items to choose from, then there will be 2^n possible combinations of items for the knapsack.
- An item is either chosen or not chosen.
- A bit string of 0's and 1's is generated which is of length n.
- If the ith symbol of a bit string is 0, then the ith item is not chosen and if it is 1, the ith item is chosen.

DRAWBACKS:

- Rarely yields efficient algorithms
- Some brute-force algorithms are unacceptably slow
- Not as constructive as some other design techniques

Complexity : O(N*2^N)

2. GREEDY ALGORITHM

- In this approach, we first sort items by worth density, in descending order.
- Then we start with the highest worth item. Next, we put items into the bag until the next item on the list cannot fit.
- Then we try to fill the remaining capacity with any of the remaining items on the list that can fit.
- The greedy algorithm doesn't improve upon the solution it returns. It simply adds the next highest worth item it can to the bag.

DRAWBACK:

● Hard to design: Once you have found the right greedy approach, designing greedy algorithms can be easy. However, finding the right approach can be hard.

● Hard to verify: Showing a greedy algorithm is correct often requires a nuanced argument.

Complexity : O(NlogN) + O(N) = O(NlogN)

3. BACKTRACKING APPROACH

The idea of backtracking is to construct solutions one component at a time and evaluate such partially constructed solutions. This partially constructed solution can be developed further without violating the problem constraints.

Algorithm:

• It is convenient to implement this kind of processing by constructing a tree of choices being made called the "State Space Tree".

• Its root represents an initial state before the search for the solution begins.

• The nodes of the first level in the tree represent the choices for the first component of a solution and the nodes of a second level represent the choices for the second component and so on.

• A node in the state space tree is promising if it corresponds to the partially constructed solution that may lead to the complete solution otherwise the nodes are called non-promising.

• Leaves of the tree represent either the non- promising dead end or complete solution found by the algorithm.

DRAWBACKS:

- One is thrashing, i.e., repeated failure due to the same reason. Thrashing occurs because the standard backtracking algorithm does not identify the real reason of the conflict, i.e., the conflicting variables. Therefore, search in different parts of the space keeps failing for the same reason. Thrashing can be avoided by intelligent backtracking, i.e., by a scheme on which backtracking is done directly to the variable that caused the failure.
- The other drawback of backtracking is having to perform redundant work. Even if the conflicting values of variables is identified during the intelligent backtracking, they are not remembered for immediate detection of the same conflict in a subsequent computation. However, there is a backtracking based method that eliminates both of the above drawbacks of backtracking. This method is traditionally called dependency-directed backtracking and is used in truth maintenance systems.
- It should be noted that using advanced techniques adds other expenses to the algorithm that has to be balanced with the overall advantage of using them.

PROPOSED METHOD:

DYNAMIC PROGRAMMING

• Firstly, a dynamic programming bag packing solution enumerates the entire solution space with all possibilities of item combinations that could be used to pack our bag. Where a greedy algorithm chooses the most optimal local solution, dynamic programming algorithms are able to find the most optimal global solution.

• Secondly, dynamic programming uses memorization to store the results of previously computed operations and returns the cached result when the operation occurs again. This allows it to "remember" previous combinations. This takes less time than it would to re-compute the answer again.

Dynamic Programming solves each of the smaller subproblems only once and records the results in a table rather than solving overlapping subproblems over and over again. The table is then used to obtain a solution to the original problem. The classical dynamic programming approach works bottom-up.

Algorithm:

• Create a matrix representing all subsets of the items (the solution space) with rows representing items and columns representing the bag's remaining weight capacity

• Loop through the matrix and calculate the worth that can be obtained by each combination of items at each stage of the bag's capacity

• Examine the completed matrix to determine which items to add to the bag in order to produce the maximum possible worth for the bag in total.

In terms of memory, Dynamic Programming requires a two-dimensional array with rows equal to the number of items and columns equal to the capacity of the knapsack. This algorithm is probably one of the easiest to implement because it does not require the use of any additional structures.

Keywords: int, for, if, new, sizeof, return, strlen.

ADVANTAGES:

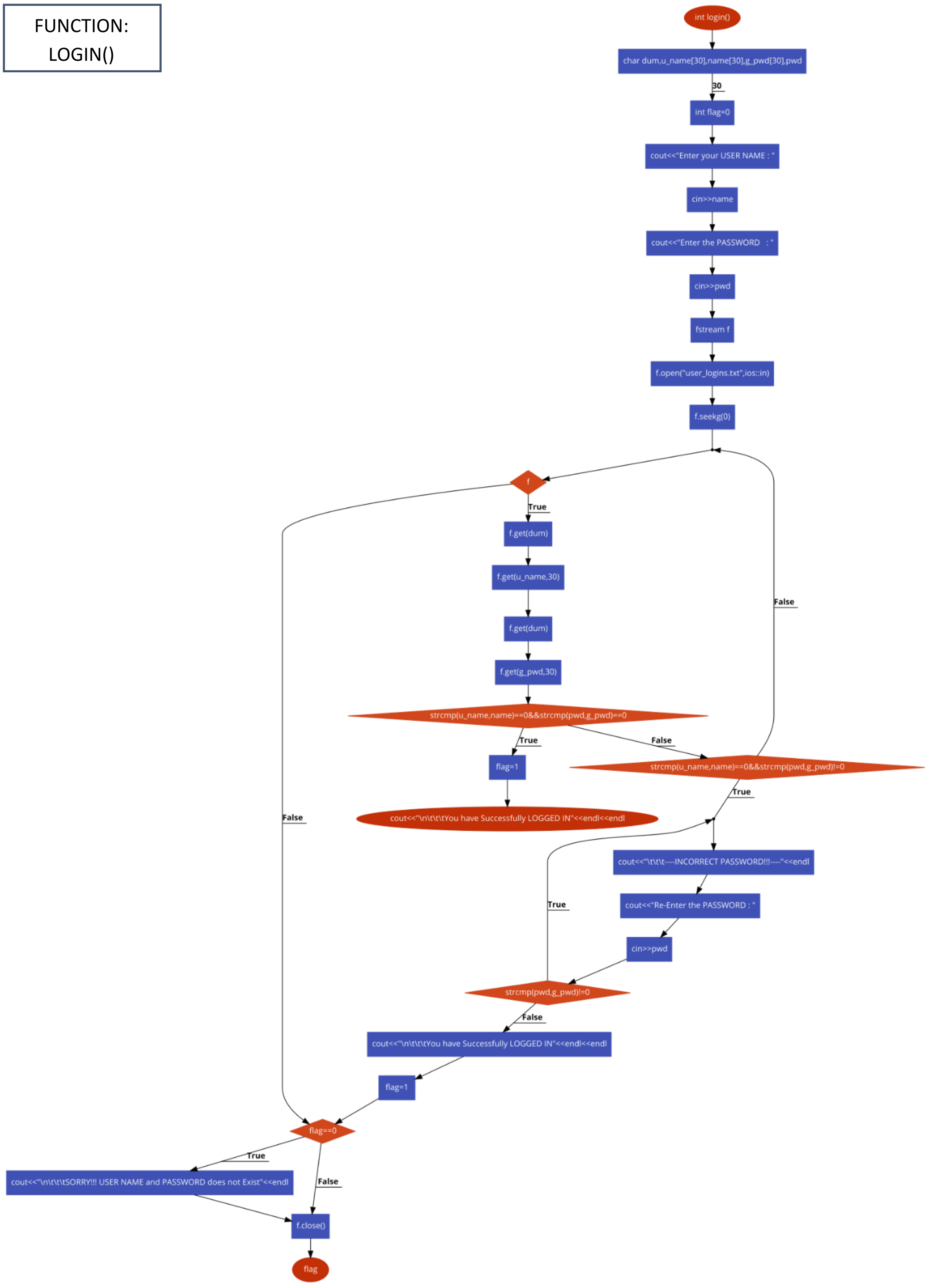
Dynamic programming means that you sort of have efficient access to already computed results for your algorithm inputs. Ideally, you may find a special order in which to perform the sub-problems computations, so that you don’t have to remember much (and also eliminate any complicated searches for the results).

So, in short, dynamic programming has the advantage of **much greater efficiency** (where the simple divide-et-impera method would re-compute sub-problems solutions).

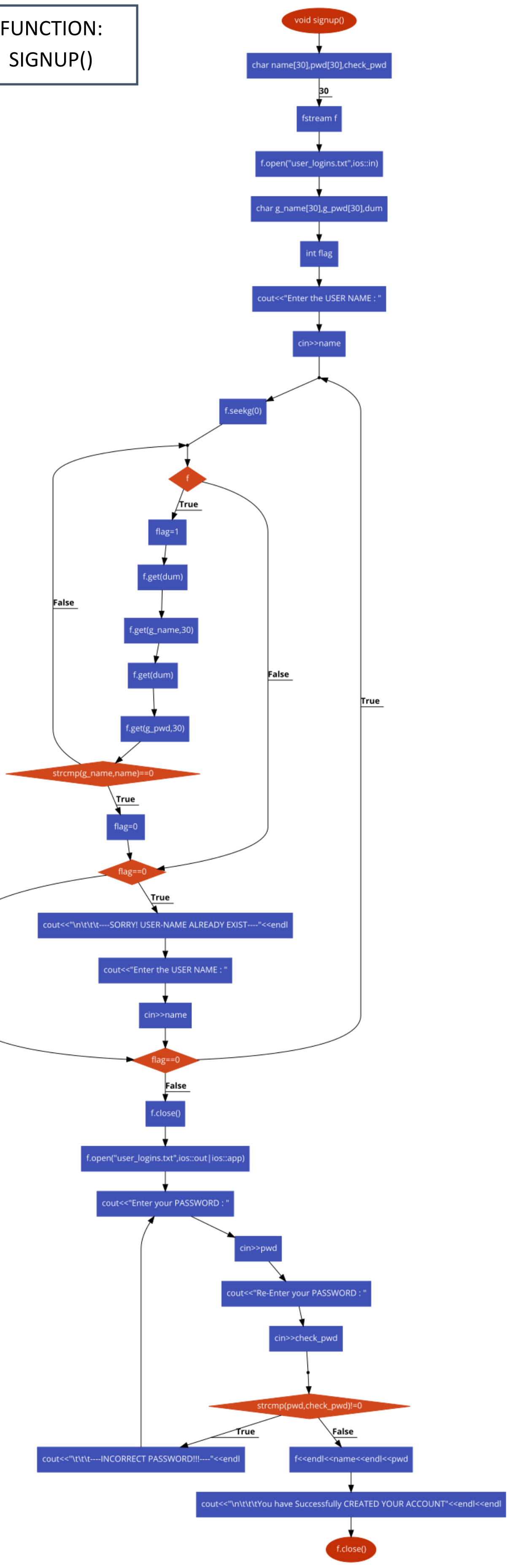
Complexity = O (N*Capacity)

PROPOSED SYSTEM ARCHITECTURE:

FUNCTION:
LOGIN()

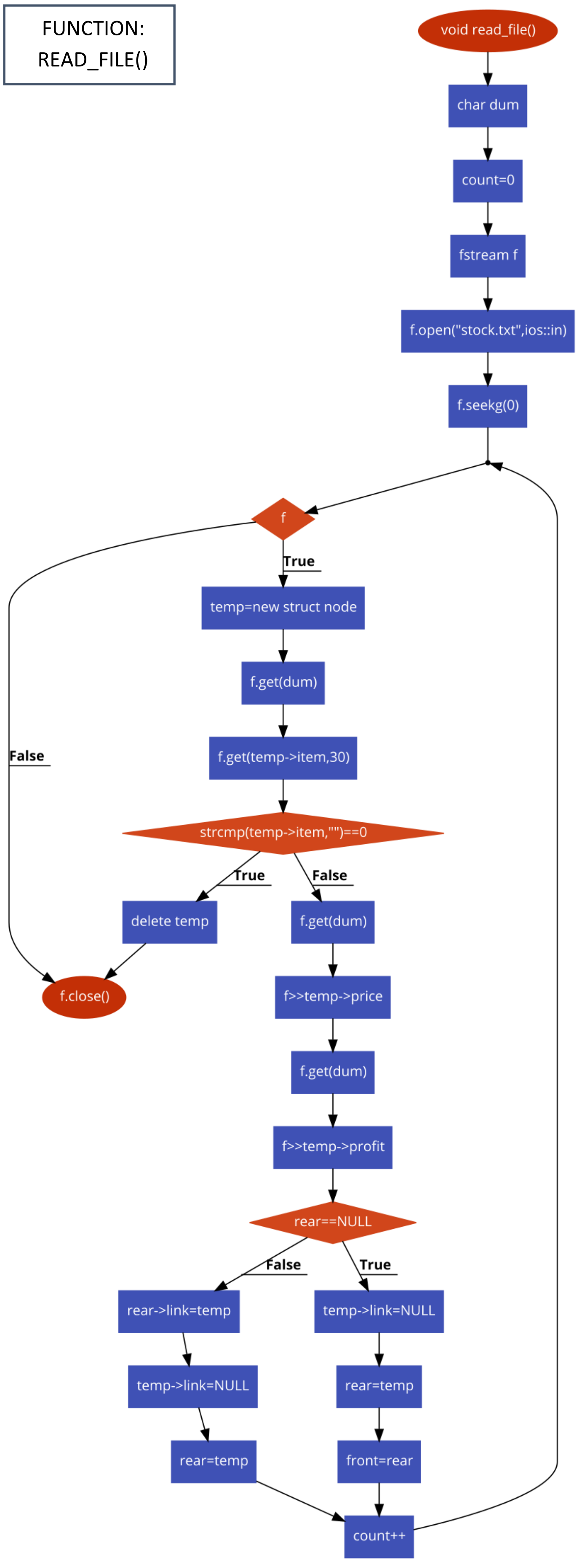

int login()
char dum,u_name[30],name[30],g_pwd[30],pwd
30
int flag=0
cout<<"Enter your USER NAME : "
cin>>name
cout<<"Enter the PASSWORD : "
cin>>pwd
fstream f
f.open("user_logins.txt",ios::in)
f.seekg(0)
f
True
f.get(dum)
f.get(u_name,30)
f.get(dum)
f.get(g_pwd,30)
False
strcmp(u_name,name)==0&&strcmp(pwd,g_pwd)==0
True
flag=1
False
strcmp(u_name,name)==0&&strcmp(pwd,g_pwd)!=0
True
cout<<"\n\t\t\tYou have Successfully LOGGED IN"<<endl<<endl
False
cout<<"\t\t\t----INCORRECT PASSWORD!!!----"<<endl
cout<<"Re-Enter the PASSWORD : "
True
cin>>pwd
strcmp(pwd,g_pwd)!=0
False
cout<<"\n\t\t\tYou have Successfully LOGGED IN"<<endl<<endl
flag=1
flag==0
True
cout<<"\n\t\t\tSORRY!!! USER NAME and PASSWORD does not Exist"<<endl
False
f.close()
flag

FUNCTION:
SIGNUP()

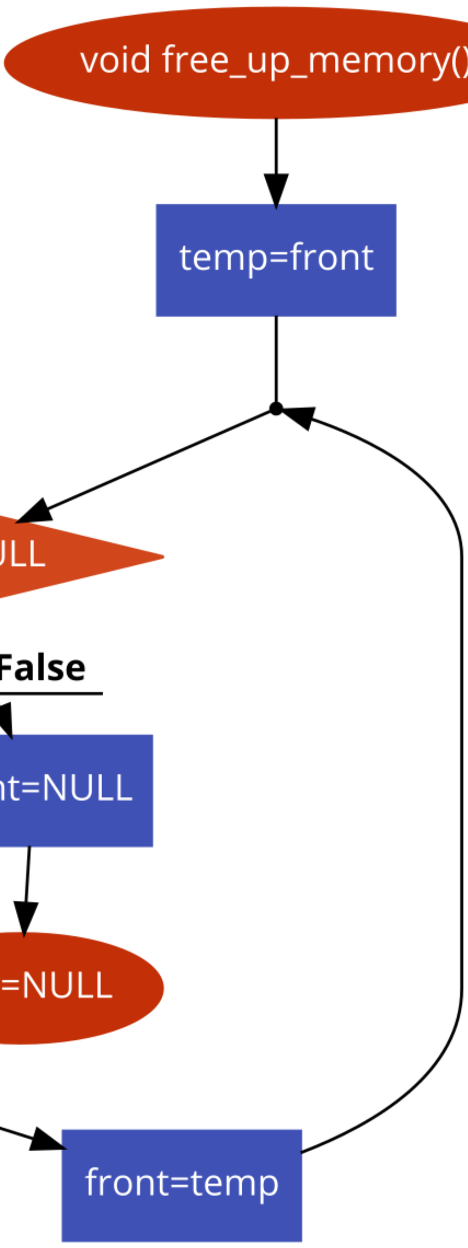


void signup()
char name[30],pwd[30],check_pwd
30
fstream f
f.open("user_logins.txt",ios::in)
char g_name[30],g_pwd[30],dum
int flag
cout<<"Enter the USER NAME : "
cin>>name
f.seekg(0)
f
True
flag=1
f.get(dum)
f.get(g_name,30)
f.get(dum)
f.get(g_pwd,30)
False
False
True
strcmp(g_name,name)==0
True
flag=0
flag==0
True
cout<<"\n\t\t\t----SORRY! USER-NAME ALREADY EXIST----"<<endl
False
cout<<"Enter the USER NAME : "
cin>>name
flag==0
False
f.close()
f.open("user_logins.txt",ios::out|ios::app)
cout<<"Enter your PASSWORD : "
cin>>pwd
cout<<"Re-Enter your PASSWORD : "
cin>>check_pwd
strcmp(pwd,check_pwd)!=0
True
False
cout<<"\t\t\t----INCORRECT PASSWORD!!!----"<<endl
f<<endl<<name<<endl<<pwd
cout<<"\n\t\t\tYou have Successfully CREATED YOUR ACCOUNT"<<endl<<endl
f.close()

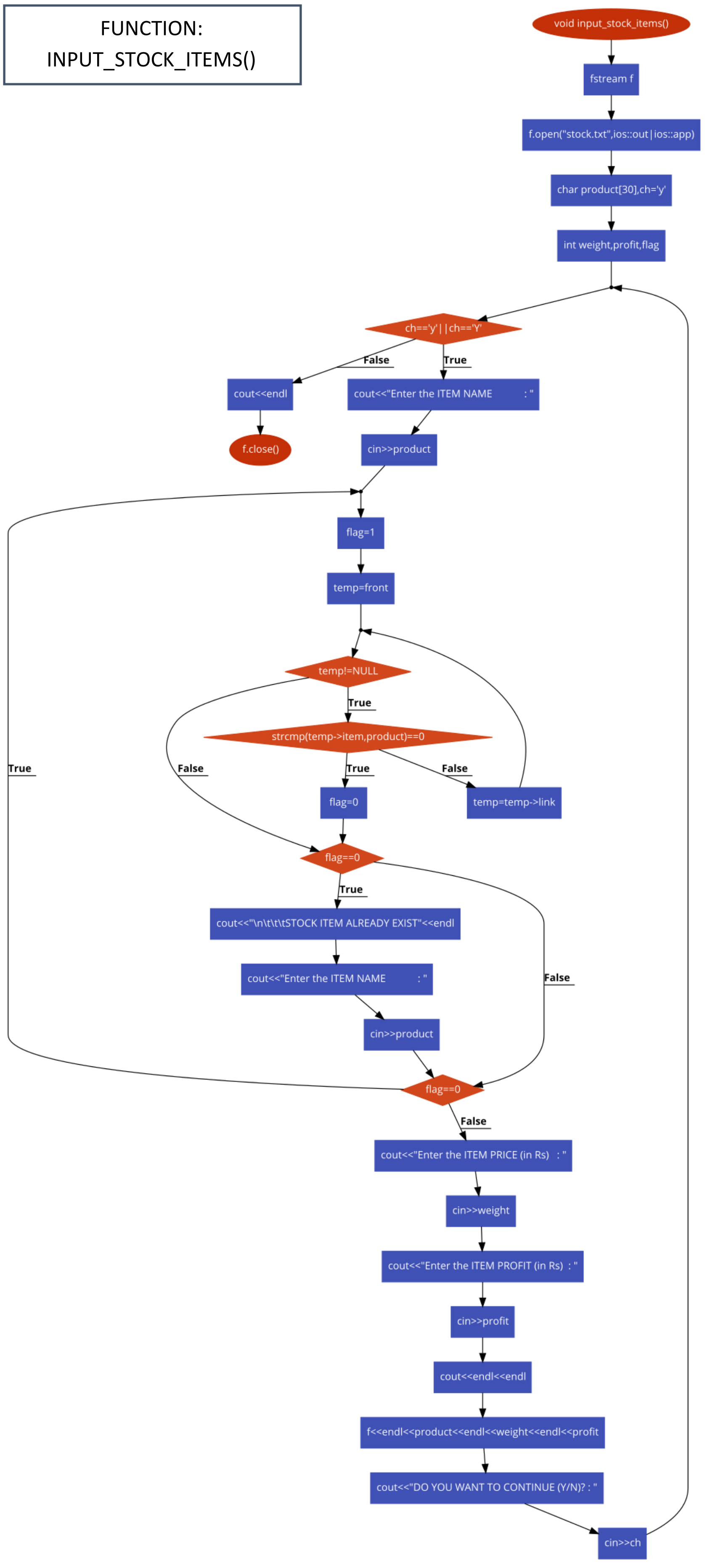
FUNCTION:
READ_FILE()



FUNCTION:
FREE_UP_MEMORY()

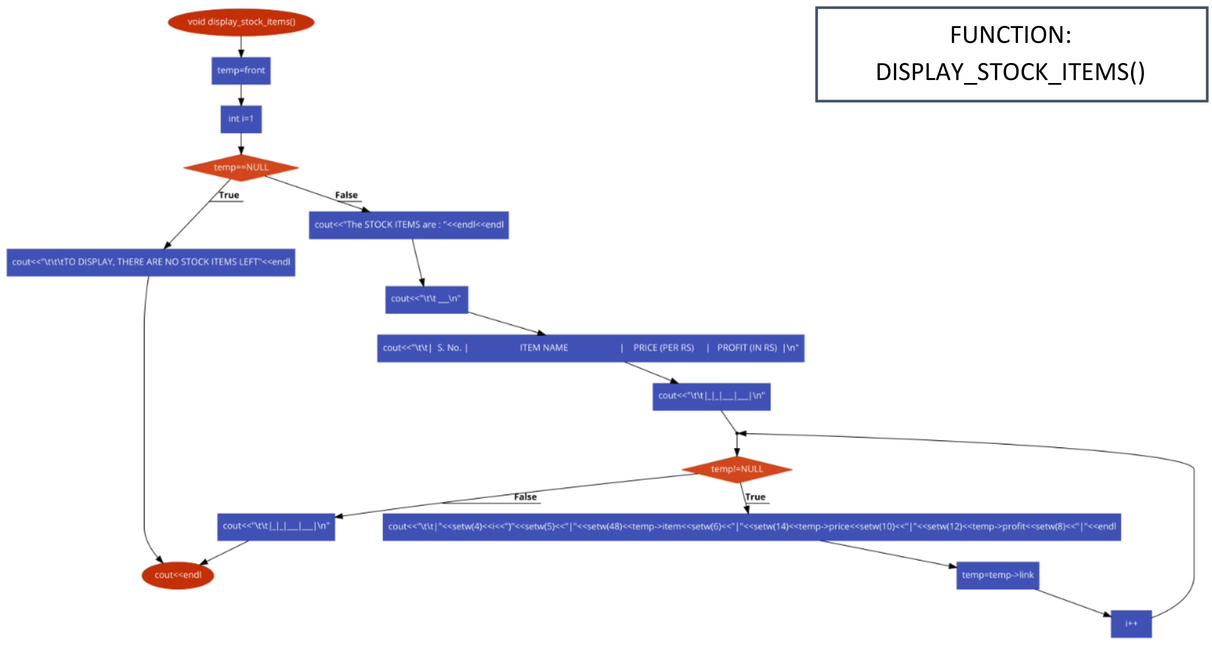


FUNCTION:
INPUT_STOCK_ITEMS()



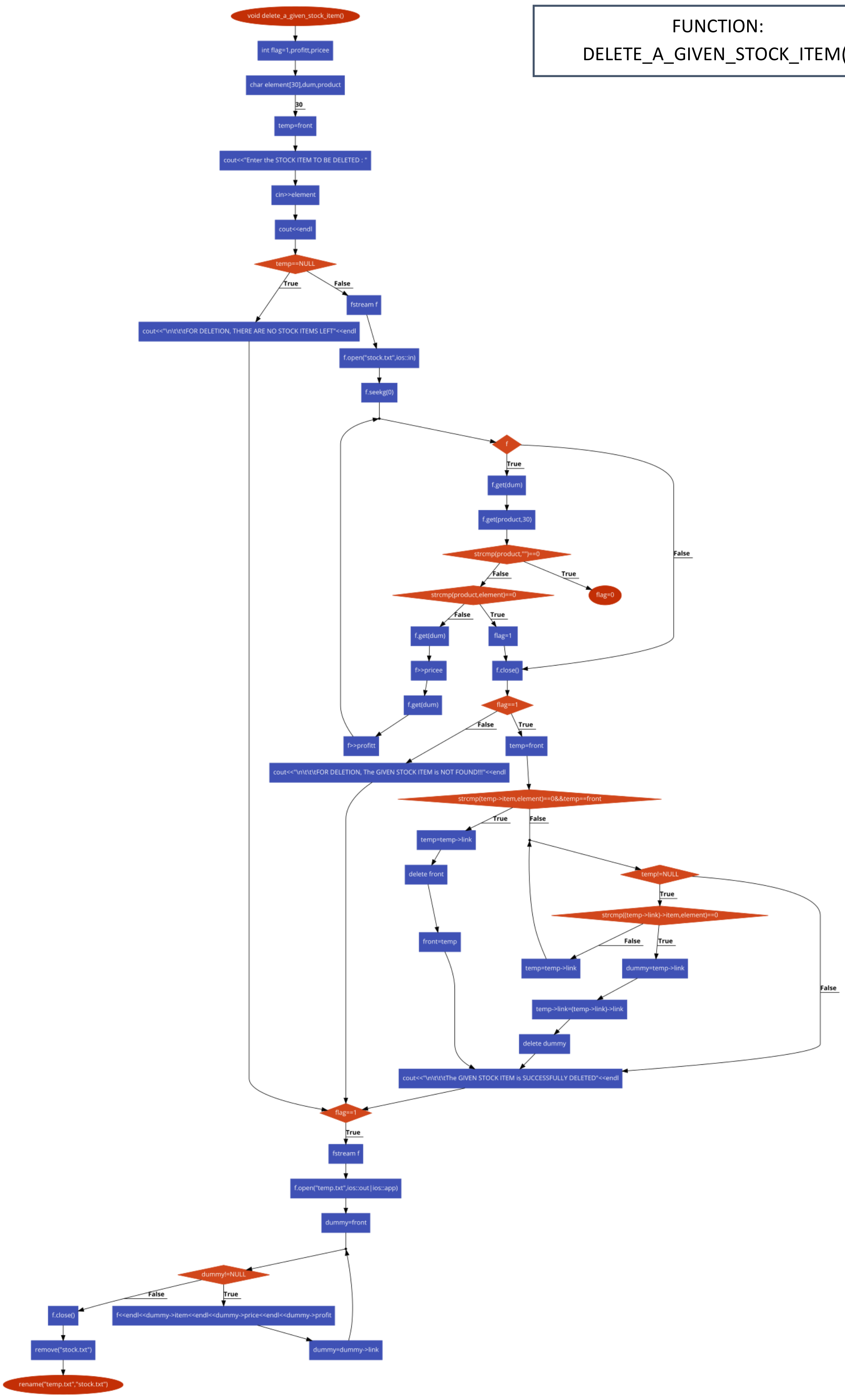
void input_stock_items()
fstream f
f.open("stock.txt",ios::out|ios::app)
char product[30],ch='y'
int weight,profit,flag
ch=='y'||ch=='Y'
False
True
cout<<endl
cout<<"Enter the ITEM NAME : "
f.close()
cin>>product
flag=1
temp=front
temp!=NULL
True
strcmp(temp->item,product)==0
True
False
True
False
flag=0
temp=temp->link
flag==0
True
cout<<"\n\t\t\tSTOCK ITEM ALREADY EXIST"<<endl
False
cout<<"Enter the ITEM NAME : "
cin>>product
flag==0
False
cout<<"Enter the ITEM PRICE (in Rs) : "
cin>>weight
cout<<"Enter the ITEM PROFIT (in Rs) : "
cin>>profit
cout<<endl<<endl
f<<endl<<product<<endl<<weight<<endl<<profit
cout<<"DO YOU WANT TO CONTINUE (Y/N)? : "
cin>>ch

FUNCTION:
DISPLAY_STOCK_ITEMS()


void display_stock_items()
temp=front
int i=1
temp==NULL
True
False
cout<<"\t\t\tTO DISPLAY, THERE ARE NO STOCK ITEMS LEFT"<<endl
cout<<"The STOCK ITEMS are : "<<endl<<endl
cout<<"\t\t __\n"
cout<<"\t\t| S. No. | ITEM NAME | PRICE (PER RS) | PROFIT (IN RS) |\n"
cout<<"\t\t|_|_|___|___|\n"
temp!=NULL
False
True
cout<<"\t\t|_|_|___|___|\n"
cout<<"\t\t|"<<setw(4)<<i<<")"<<setw(5)<<"|"<<setw(48)<<temp->item<<setw(6)<<"|"<<setw(14)<<temp->price<<setw(10)<<"|"<<setw(12)<<temp->profit<<setw(8)<<"|"<<endl
temp=temp->link
i++
cout<<endl

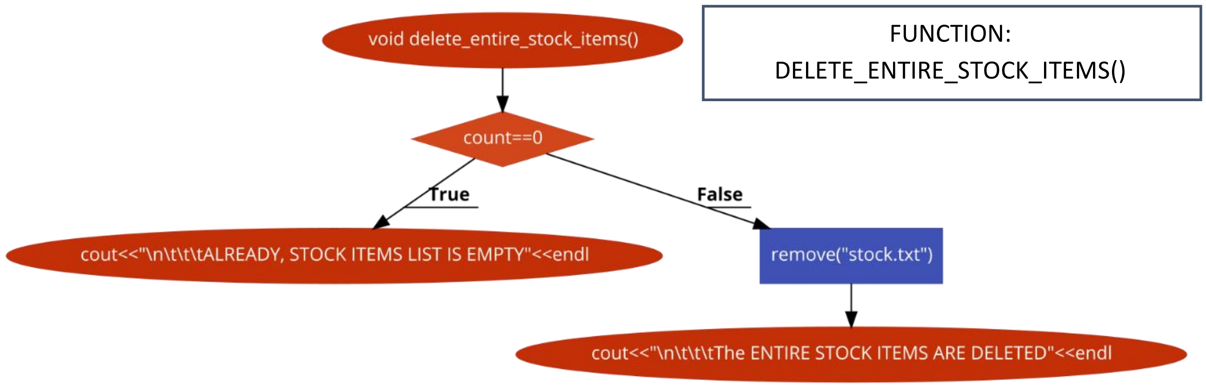
FUNCTION:

DELETE_A_GIVEN_STOCK_ITEM()

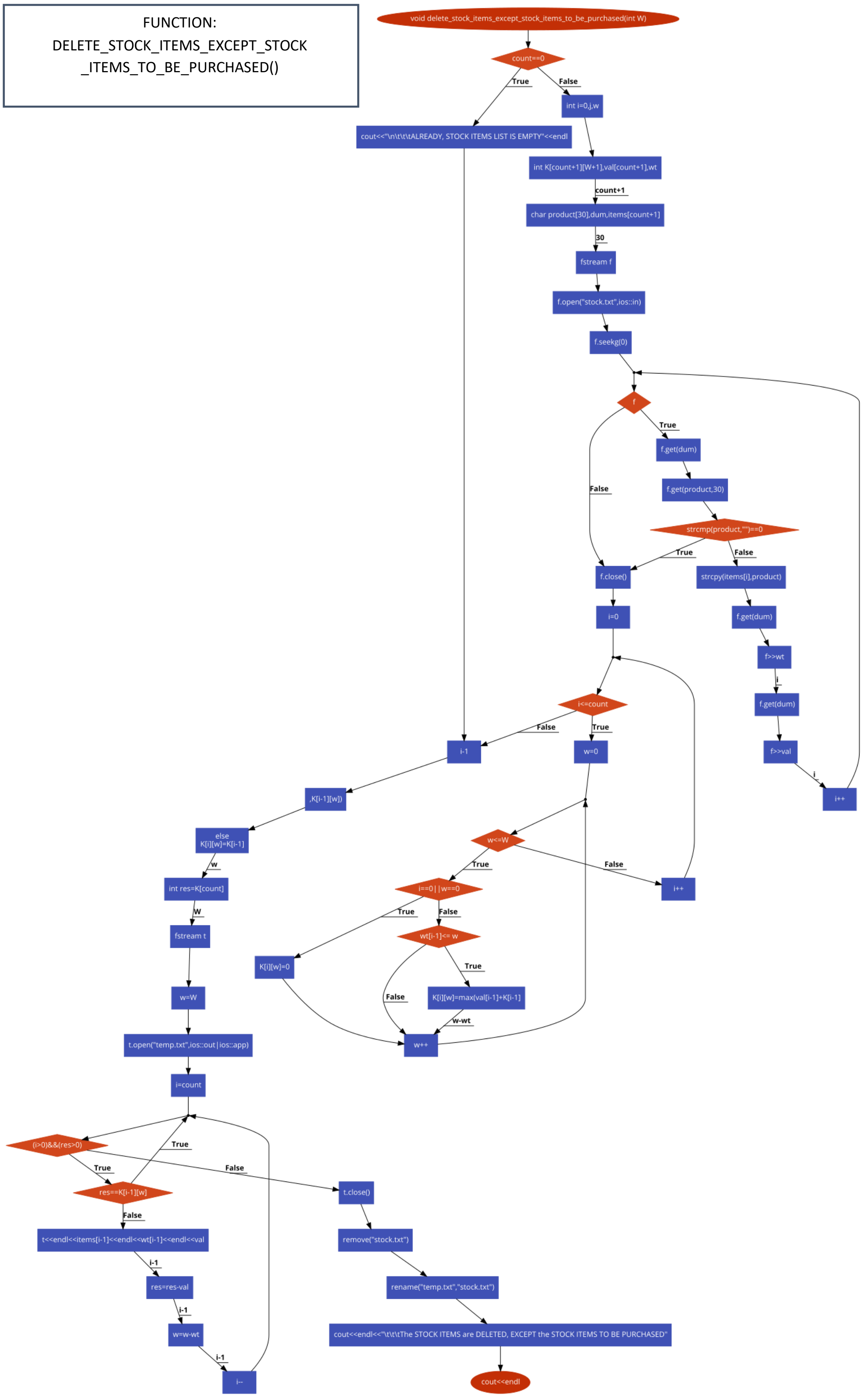


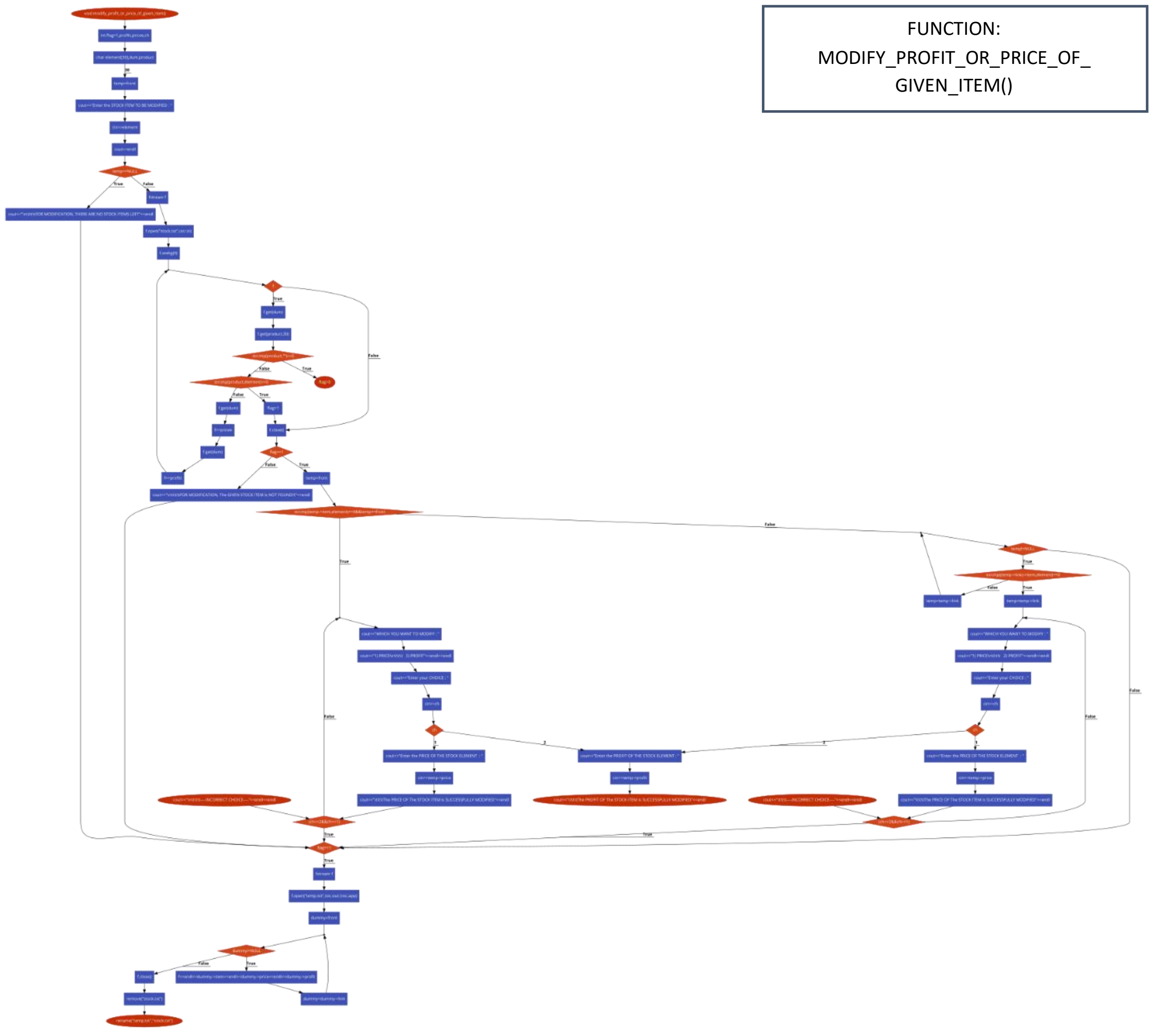
void delete_a_given_stock_item()
int flag=1,profitt,pricee
char element[30],dum,product
30
temp=front
cout<<"Enter the STOCK ITEM TO BE DELETED : "
cin>>element
cout<<endl
temp==NULL
True
False
cout<<"\n\t\t\tFOR DELETION, THERE ARE NO STOCK ITEMS LEFT"<<endl
fstream f
f.open("stock.txt",ios::in)
f.seekg(0)
f
True
False
f.get(dum)
f.get(product,30)
strcmp(product,"")==0
False
True
flag=0
strcmp(product,element)==0
False
True
f.get(dum)
flag=1
f>>pricee
f.close()
f.get(dum)
flag==1
False
True
f>>profitt
temp=front
cout<<"\n\t\t\tFOR DELETION, The GIVEN STOCK ITEM is NOT FOUND!!!"<<endl
strcmp(temp->item,element)==0&&temp==front
True
False
temp=temp->link
delete front
temp!=NULL
True
strcmp((temp->link)->item,element)==0
False
True
front=temp
temp=temp->link
dummy=temp->link
False
temp->link=(temp->link)->link
delete dummy
cout<<"\n\t\t\tThe GIVEN STOCK ITEM is SUCCESSFULLY DELETED"<<endl
flag==1
True
fstream f
f.open("temp.txt",ios::out|ios::app)
dummy=front
dummy!=NULL
False
True
f.close()
f<<endl<<dummy->item<<endl<<dummy->price<<endl<<dummy->profit
remove("stock.txt")
dummy=dummy->link
rename("temp.txt","stock.txt")

FUNCTION:
DELETE_ENTIRE_STOCK_ITEMS()

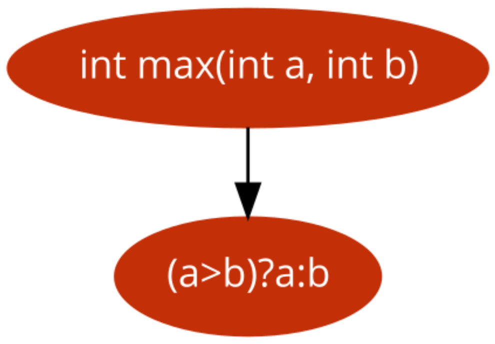

void delete_entire_stock_items()
count==0
True
False
cout<<"\n\t\t\tALREADY, STOCK ITEMS LIST IS EMPTY"<<endl
remove("stock.txt")
cout<<"\n\t\t\tThe ENTIRE STOCK ITEMS ARE DELETED"<<endl

FUNCTION:
DELETE_STOCK_ITEMS_EXCEPT_STOCK_ITEMS_TO_BE_PURCHASED()



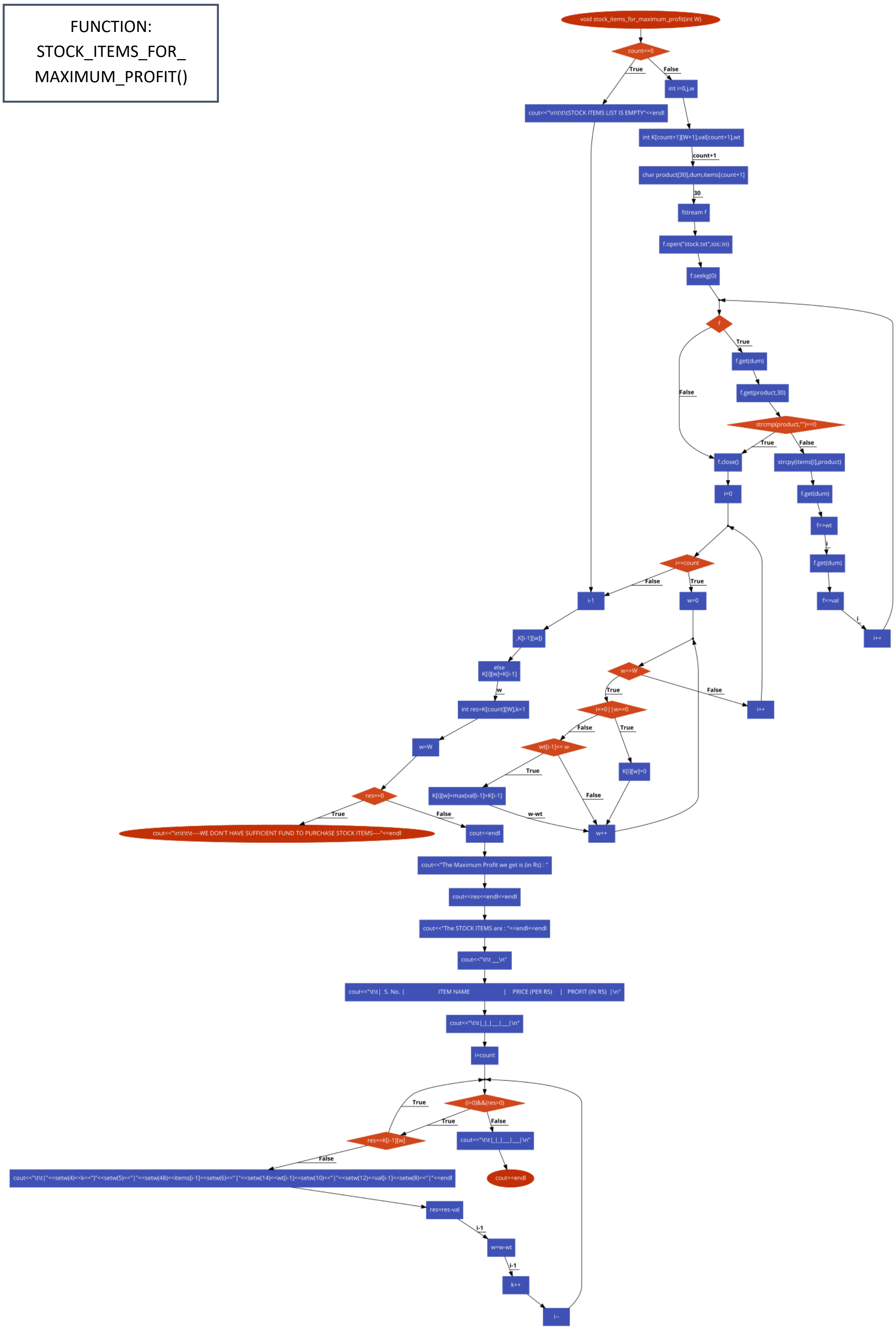


FUNCTION:
MODIFY_PROFIT_OR_PRICE_OF_
GIVEN_ITEM()

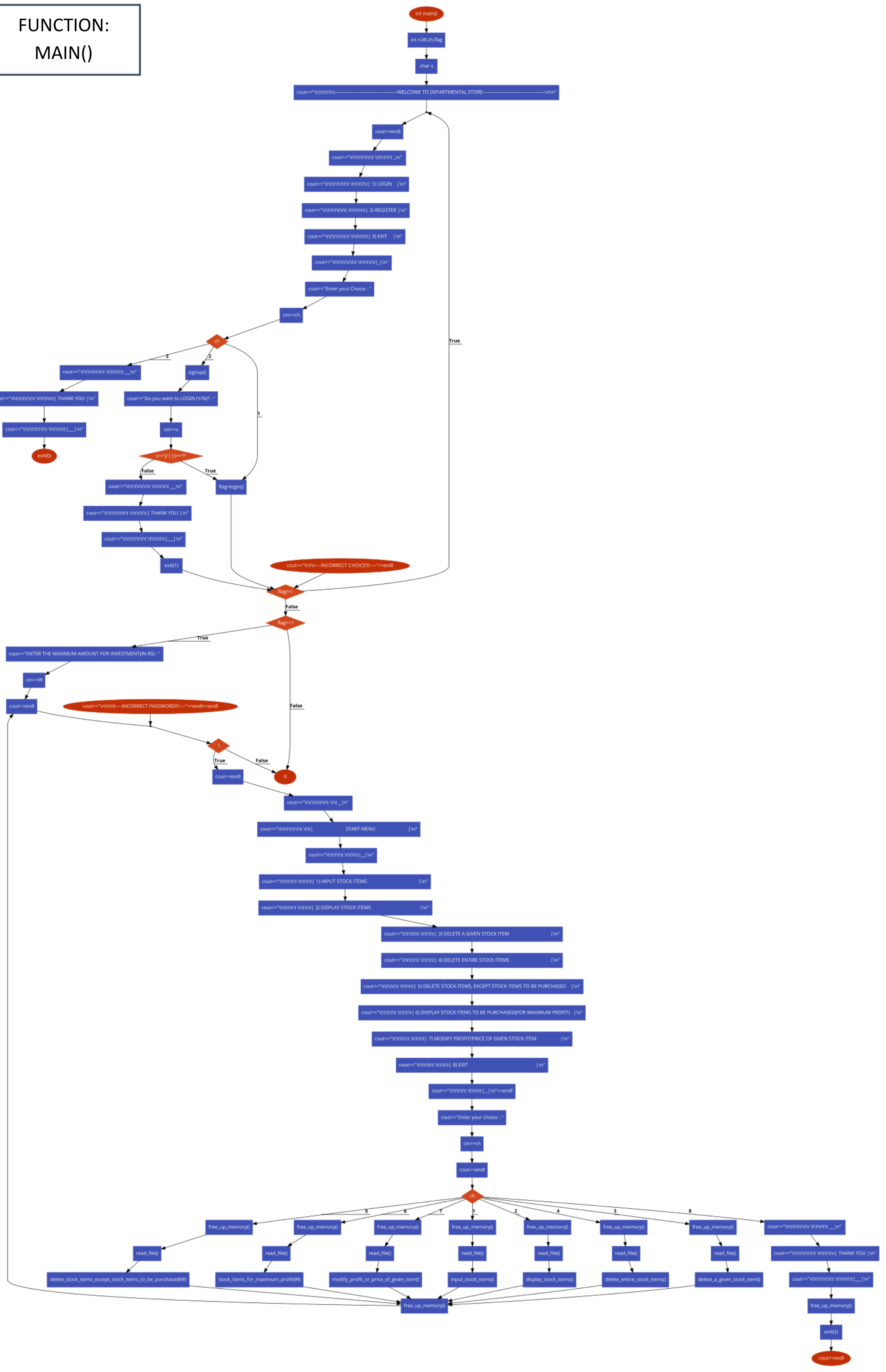


FUNCTION:
MAX()

FUNCTION:
STOCK_ITEMS_FOR_
MAXIMUM_PROFIT()


void stock_items_for_maximum_profit(int W)
count==0
True
False
cout<<"\n\t\t\tSTOCK ITEMS LIST IS EMPTY"<<endl
int i=0,j,w
int K[count+1][W+1],val[count+1],wt
count+1
char product[30],dum,items[count+1]
30
fstream f
f.open("stock.txt",ios::in)
f.seekg(0)
f
True
False
f.get(dum)
f.get(product,30)
strcmp(product,"")==0
True
False
f.close()
strcpy(items[i],product)
i=0
f.get(dum)
f>>wt
i
f.get(dum)
f>>val
i
i++
i<=count
False
True
i-1
w=0
,K[i-1][w])
else
K[i][w]=K[i-1]
w
int res=K[count][W],k=1
w<=W
True
False
i++
i==0||w==0
False
True
wt[i-1]<= w
True
False
K[i][w]=0
K[i][w]=max(val[i-1]+K[i-1]
w-wt
w++
w=W
res==0
True
False
cout<<"\n\t\t\t----WE DON'T HAVE SUFFICIENT FUND TO PURCHASE STOCK ITEMS----"<<endl
cout<<endl
cout<<"The Maximum Profit we get is (in Rs) : "
cout<<res<<endl<<endl
cout<<"The STOCK ITEMS are : "<<endl<<endl
cout<<"\t\t __\n"
cout<<"\t\t| S. No. | ITEM NAME | PRICE (PER RS) | PROFIT (IN RS) |\n"
cout<<"\t\t|_|_|___|___|\n"
i=count
(i>0)&&(res>0)
True
True
False
res==K[i-1][w]
False
cout<<"\t\t|_|_|___|___|\n"
cout<<"\t\t|"<<setw(4)<<k<<")"<<setw(5)<<"|"<<setw(48)<<items[i-1]<<setw(6)<<"|"<<setw(14)<<wt[i-1]<<setw(10)<<"|"<<setw(12)<<val[i-1]<<setw(8)<<"|"<<endl
cout<<endl
res=res-val
i-1
w=w-wt
i-1
k++
i--

FUNCTION:
MAIN()

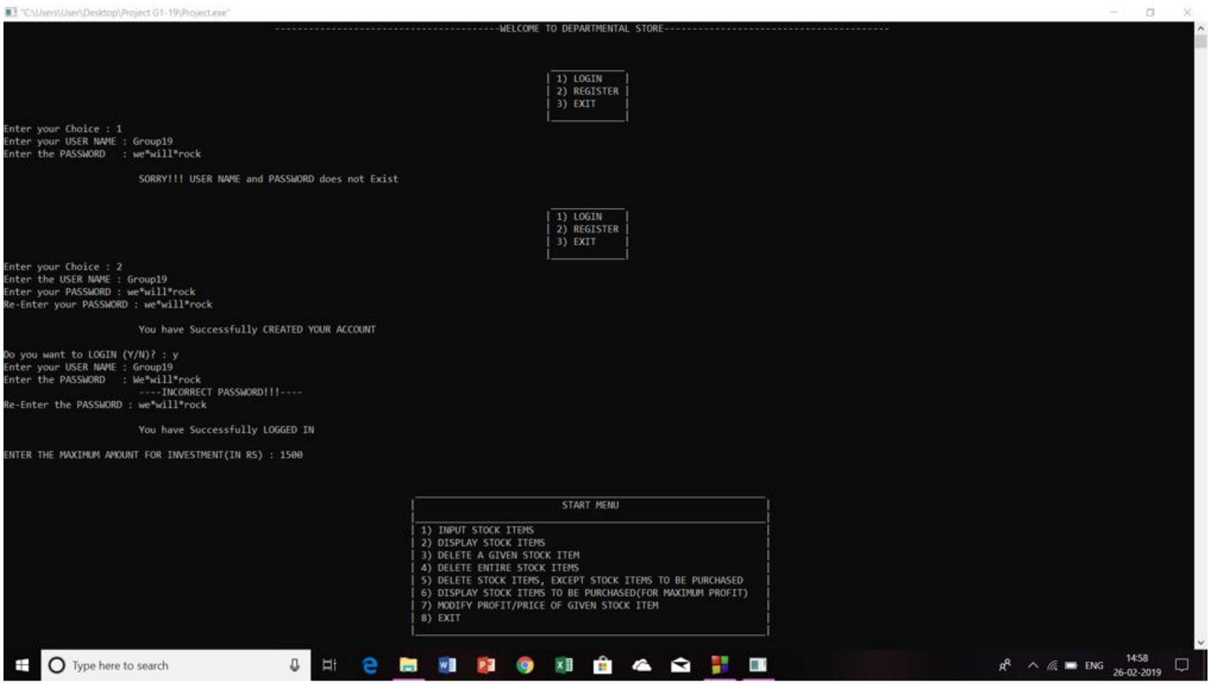


int main()
int n,W,ch,flag
char s
cout<<"\t\t\t\t\t\t---WELCOME TO DEPARTMENTAL STORE---\n\n"
cout<<endl
cout<<"\t\t\t\t\t\t\t \t\t\t\t\t _\n"
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t| 1) LOGIN |\n"
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t| 2) REGISTER |\n"
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t| 3) EXIT |\n"
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t|_|\n"
cout<<"Enter your Choice : "
cin>>ch
ch
3
2
1
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t __\n"
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t| THANK YOU |\n"
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t|__|\n"
exit(0)
signup()
cout<<"Do you want to LOGIN (Y/N)? : "
cin>>s
s=='y' | | s=='Y'
False
True
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t __\n"
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t| THANK YOU |\n"
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t|__|\n"
exit(1)
flag=login()
cout<<"\t\t\t\t----INCORRECT CHOICE!!!----"<<endl
True
flag!=1
False
flag==1
True
False
cout<<"ENTER THE MAXIMUM AMOUNT FOR INVESTMENT(IN RS) : "
cin>>W
cout<<endl
cout<<"\n\t\t\t\t----INCORRECT PASSWORD!!!----"<<endl<<endl
1
True
False
0
cout<<endl
cout<<"\t\t\t\t\t\t\t \t\t\t _\n"
cout<<"\t\t\t\t\t\t\t \t\t\t| START MENU |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t|__|\n"
cout<<"\t\t\t\t\t\t \t\t\t\t| 1) INPUT STOCK ITEMS |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t| 2) DISPLAY STOCK ITEMS |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t| 3) DELETE A GIVEN STOCK ITEM |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t| 4) DELETE ENTIRE STOCK ITEMS |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t| 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t| 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT) |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t| 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t| 8) EXIT |\n"
cout<<"\t\t\t\t\t\t \t\t\t\t|__|\n"<<endl
cout<<"Enter your choice : "
cin>>ch
cout<<endl
ch
5
6
7
1
2
4
3
8
free_up_memory()
free_up_memory()
free_up_memory()
free_up_memory()
free_up_memory()
free_up_memory()
free_up_memory()
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t __\n"
read_file()
read_file()
read_file()
read_file()
read_file()
read_file()
read_file()
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t| THANK YOU |\n"
delete_stock_items_except_stock_items_to_be_purchased(W)
stock_items_for_maximum_profit(W)
modify_profit_or_price_of_given_item()
input_stock_items()
display_stock_items()
delete_entire_stock_items()
delete_a_given_stock_item()
cout<<"\t\t\t\t\t\t\t\t \t\t\t\t\t|__|\n"
free_up_memory()
free_up_memory()
exit(2)
cout<<endl

IMPLEMENTATION DETAILS

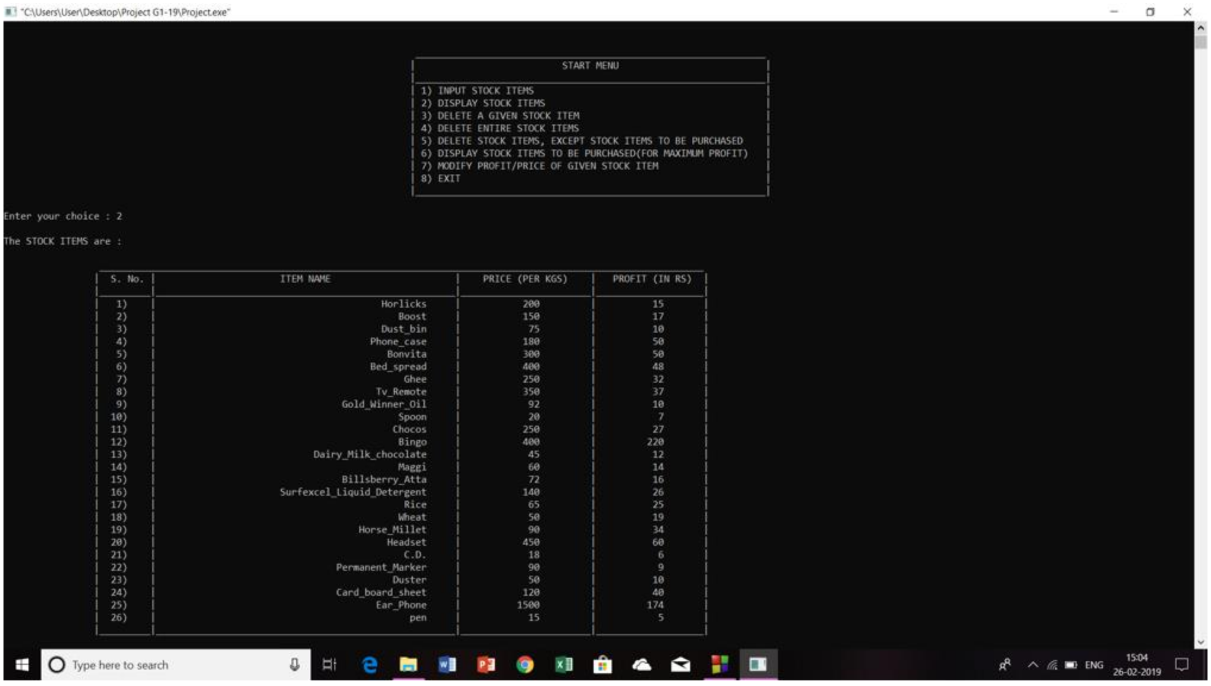
Please refer to my GitHub Repository.

TESTING AND SAMPLE SCREENSHOTS OF OUTPUT:



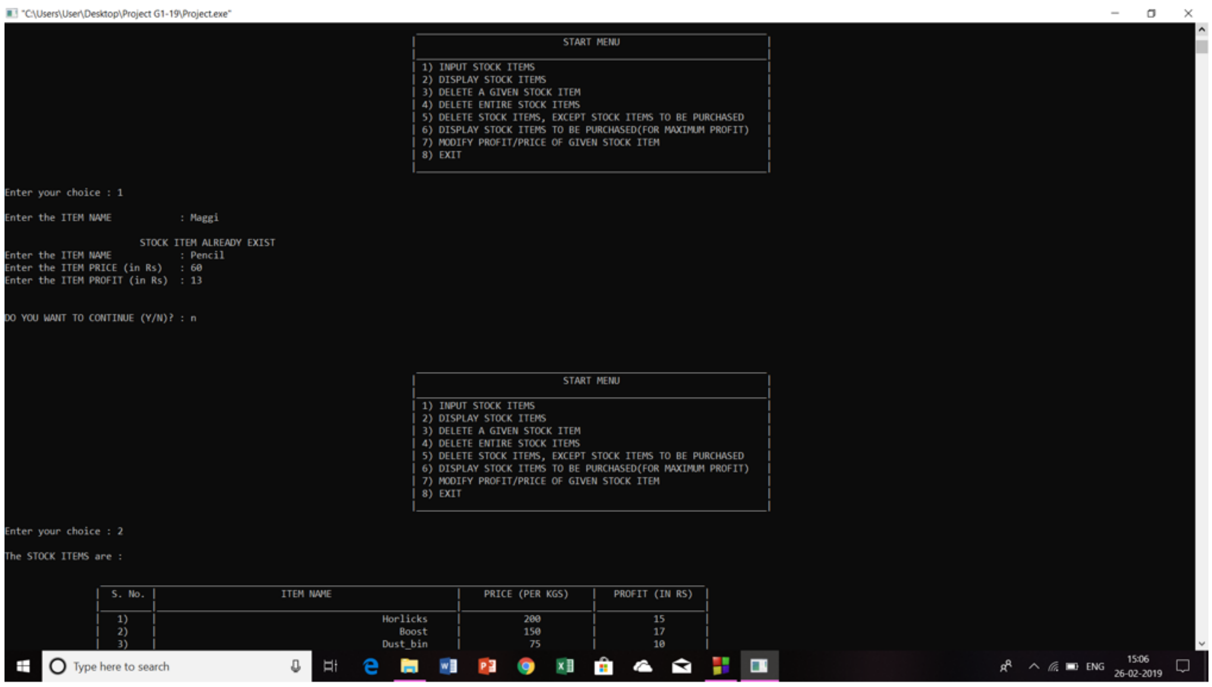
"C:\Users\User\Desktop\Project G1-19\Project.exe"

```
                                        ----------------------------------------WELCOME TO DEPARTMENTAL STORE----------------------------------------


                                                                                                    | 1) LOGIN    |
                                                                                                    | 2) REGISTER |
                                                                                                    | 3) EXIT     |
                                                                                                    |_____________|

Enter your Choice : 1
Enter your USER NAME : Group19
Enter the PASSWORD   : we*will*rock

                        SORRY!!! USER NAME and PASSWORD does not Exist


                                                                                                    | 1) LOGIN    |
                                                                                                    | 2) REGISTER |
                                                                                                    | 3) EXIT     |
                                                                                                    |_____________|

Enter your Choice : 2
Enter the USER NAME : Group19
Enter your PASSWORD : we*will*rock
Re-Enter your PASSWORD : we*will*rock

                        You have Successfully CREATED YOUR ACCOUNT

Do you want to LOGIN (Y/N)? : y
Enter your USER NAME : Group19
Enter the PASSWORD   : We*will*rock
                        ----INCORRECT PASSWORD!!!----
Re-Enter the PASSWORD : we*will*rock

                        You have Successfully LOGGED IN

ENTER THE MAXIMUM AMOUNT FOR INVESTMENT(IN RS) : 1500


                                                                          ________________________________________________________________
                                                                          |                           START MENU                           |
                                                                          |________________________________________________________________|
                                                                          | 1) INPUT STOCK ITEMS                                           |
                                                                          | 2) DISPLAY STOCK ITEMS                                         |
                                                                          | 3) DELETE A GIVEN STOCK ITEM                                   |
                                                                          | 4) DELETE ENTIRE STOCK ITEMS                                   |
                                                                          | 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED      |
                                                                          | 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT)     |
                                                                          | 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM                     |
                                                                          | 8) EXIT                                                        |
                                                                          |________________________________________________________________|
```



"C:\Users\User\Desktop\Project G1-19\Project.exe"

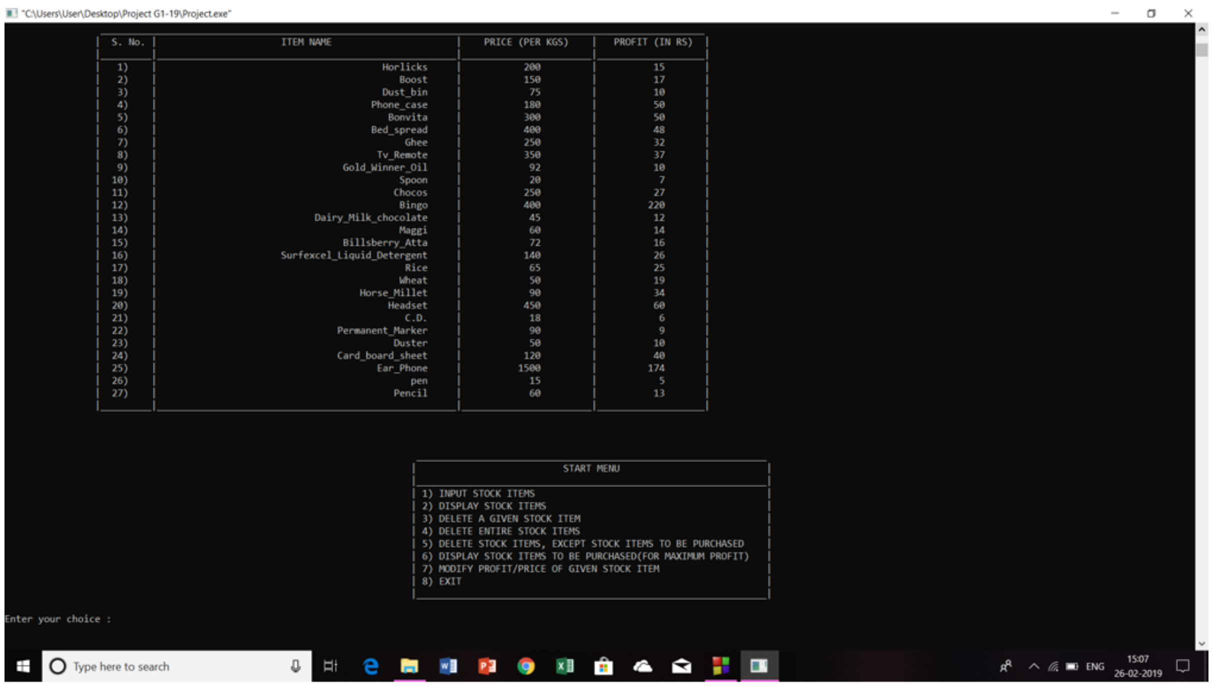
```
                                                                          ________________________________________________________________
                                                                          |                           START MENU                           |
                                                                          |________________________________________________________________|
                                                                          | 1) INPUT STOCK ITEMS                                           |
                                                                          | 2) DISPLAY STOCK ITEMS                                         |
                                                                          | 3) DELETE A GIVEN STOCK ITEM                                   |
                                                                          | 4) DELETE ENTIRE STOCK ITEMS                                   |
                                                                          | 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED      |
                                                                          | 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT)     |
                                                                          | 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM                     |
                                                                          | 8) EXIT                                                        |
                                                                          |________________________________________________________________|

Enter your choice : 2

The STOCK ITEMS are :

                 ____________________________________________________________________________________________________________
                 | S. No. |                  ITEM NAME                          |  PRICE (PER KGS)  |  PROFIT (IN RS)  |
                 |________|_____________________________________________________|___________________|__________________|
                 |   1)   |                                            Horlicks |        200        |        15        |
                 |   2)   |                                               Boost |        150        |        17        |
                 |   3)   |                                            Dust_bin |         75        |        10        |
                 |   4)   |                                          Phone_case |        180        |        50        |
                 |   5)   |                                             Bonvita |        300        |        50        |
                 |   6)   |                                          Bed_spread |        400        |        48        |
                 |   7)   |                                                Ghee |        250        |        32        |
                 |   8)   |                                           Tv_Remote |        350        |        37        |
                 |   9)   |                                     Gold_Winner_Oil |         92        |        10        |
                 |  10)   |                                               Spoon |         20        |         7        |
                 |  11)   |                                              Chocos |        250        |        27        |
                 |  12)   |                                               Bingo |        400        |       220        |
                 |  13)   |                               Dairy_Milk_chocolate |         45        |        12        |
                 |  14)   |                                               Maggi |         60        |        14        |
                 |  15)   |                                    Billsberry_Atta |         72        |        16        |
                 |  16)   |                         Surfexcel_Liquid_Detergent |        140        |        26        |
                 |  17)   |                                                Rice |         65        |        25        |
                 |  18)   |                                               Wheat |         50        |        19        |
                 |  19)   |                                        Horse_Millet |         90        |        34        |
                 |  20)   |                                             Headset |        450        |        60        |
                 |  21)   |                                                C.D. |         18        |         6        |
                 |  22)   |                                    Permanent_Marker |         90        |         9        |
                 |  23)   |                                              Duster |         50        |        10        |
                 |  24)   |                                     Card_board_sheet |        120        |        40        |
                 |  25)   |                                           Ear_Phone |       1500        |       174        |
                 |  26)   |                                                 pen |         15        |         5        |
                 |________|_____________________________________________________|___________________|__________________|
```


"C:\Users\User\Desktop\Project G1-19\Project.exe"

```
                                                      ___________________________________________________________
                                                     |                        START MENU                         |
                                                     |___________________________________________________________|
                                                     | 1) INPUT STOCK ITEMS                                      |
                                                     | 2) DISPLAY STOCK ITEMS                                    |
                                                     | 3) DELETE A GIVEN STOCK ITEM                              |
                                                     | 4) DELETE ENTIRE STOCK ITEMS                              |
                                                     | 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED |
                                                     | 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT)|
                                                     | 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM                |
                                                     | 8) EXIT                                                   |
                                                     |___________________________________________________________|

Enter your choice : 1

Enter the ITEM NAME            : Maggi

                        STOCK ITEM ALREADY EXIST
Enter the ITEM NAME            : Pencil
Enter the ITEM PRICE (in Rs)   : 60
Enter the ITEM PROFIT (in Rs)  : 13


DO YOU WANT TO CONTINUE (Y/N)? : n



                                                      ___________________________________________________________
                                                     |                        START MENU                         |
                                                     |___________________________________________________________|
                                                     | 1) INPUT STOCK ITEMS                                      |
                                                     | 2) DISPLAY STOCK ITEMS                                    |
                                                     | 3) DELETE A GIVEN STOCK ITEM                              |
                                                     | 4) DELETE ENTIRE STOCK ITEMS                              |
                                                     | 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED |
                                                     | 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT)|
                                                     | 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM                |
                                                     | 8) EXIT                                                   |
                                                     |___________________________________________________________|

Enter your choice : 2

The STOCK ITEMS are :

                 ____________________________________________________________________________________________________
                | S. No. |                  ITEM NAME                   |  PRICE (PER KGS)  |  PROFIT (IN RS)  |
                |________|______________________________________________|___________________|__________________|
                |   1)   |                               Horlicks       |       200         |        15        |
                |   2)   |                                  Boost       |       150         |        17        |
                |   3)   |                               Dust_bin       |        75         |        10        |
```

Type here to search

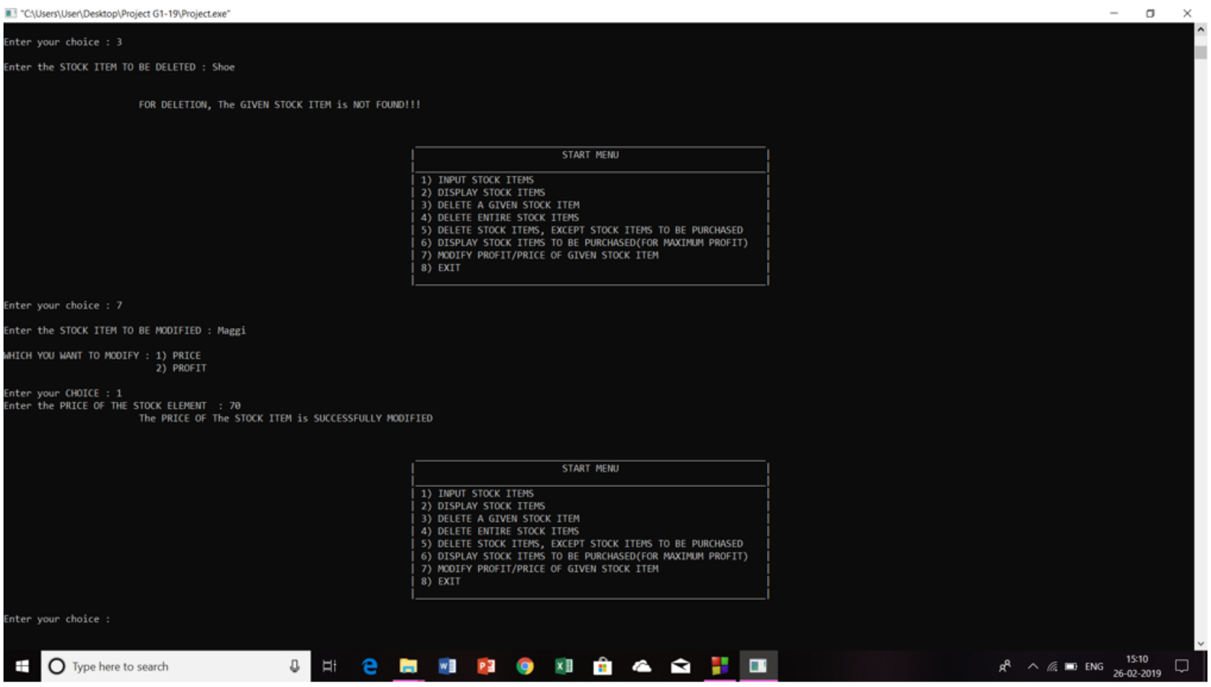
ENG 15:06 26-02-2019


"C:\Users\User\Desktop\Project G1-19\Project.exe"

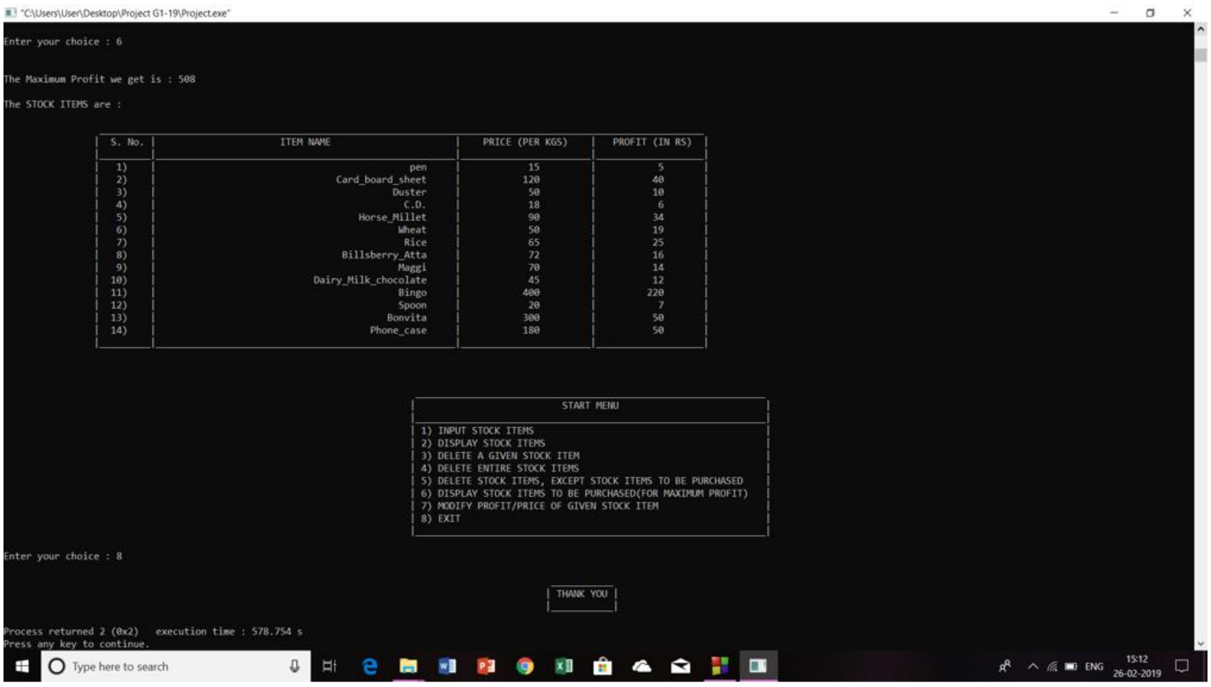
```
                 ____________________________________________________________________________________________________
                | S. No. |                  ITEM NAME                   |  PRICE (PER KGS)  |  PROFIT (IN RS)  |
                |________|______________________________________________|___________________|__________________|
                |   1)   |                               Horlicks       |       200         |        15        |
                |   2)   |                                  Boost       |       150         |        17        |
                |   3)   |                               Dust_bin       |        75         |        10        |
                |   4)   |                             Phone_case       |       180         |        50        |
                |   5)   |                                Bonvita       |       300         |        50        |
                |   6)   |                             Bed_spread       |       400         |        48        |
                |   7)   |                                   Ghee       |       250         |        32        |
                |   8)   |                              Tv_Remote       |       350         |        37        |
                |   9)   |                        Gold_Winner_Oil       |        92         |        10        |
                |  10)   |                                  Spoon       |        20         |         7        |
                |  11)   |                                 Chocos       |       250         |        27        |
                |  12)   |                                  Bingo       |       400         |       220        |
                |  13)   |                   Dairy_Milk_chocolate       |        45         |        12        |
                |  14)   |                                  Maggi       |        60         |        14        |
                |  15)   |                        Billsberry_Atta       |        72         |        16        |
                |  16)   |             Surfexcel_Liquid_Detergent       |       140         |        26        |
                |  17)   |                                   Rice       |        65         |        25        |
                |  18)   |                                  Wheat       |        50         |        19        |
                |  19)   |                           Horse_Millet       |        90         |        34        |
                |  20)   |                                Headset       |       450         |        60        |
                |  21)   |                                   C.D.       |        18         |         6        |
                |  22)   |                       Permanent_Marker       |        90         |         9        |
                |  23)   |                                 Duster       |        50         |        10        |
                |  24)   |                       Card_board_sheet       |       120         |        40        |
                |  25)   |                              Ear_Phone       |      1500         |       174        |
                |  26)   |                                    pen       |        15         |         5        |
                |  27)   |                                 Pencil       |        60         |        13        |
                |________|______________________________________________|___________________|__________________|


                                                      ___________________________________________________________
                                                     |                        START MENU                         |
                                                     |___________________________________________________________|
                                                     | 1) INPUT STOCK ITEMS                                      |
                                                     | 2) DISPLAY STOCK ITEMS                                    |
                                                     | 3) DELETE A GIVEN STOCK ITEM                              |
                                                     | 4) DELETE ENTIRE STOCK ITEMS                              |
                                                     | 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED |
                                                     | 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT)|
                                                     | 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM                |
                                                     | 8) EXIT                                                   |
                                                     |___________________________________________________________|

Enter your choice :
```

Type here to search

ENG 15:07 26-02-2019



```
Enter your choice : 3

Enter the STOCK ITEM TO BE DELETED : Shoe


                        FOR DELETION, The GIVEN STOCK ITEM is NOT FOUND!!!


                                                                         ________________________________________________________________
                                                                        |                          START MENU                            |
                                                                        |________________________________________________________________|
                                                                        | 1) INPUT STOCK ITEMS                                           |
                                                                        | 2) DISPLAY STOCK ITEMS                                         |
                                                                        | 3) DELETE A GIVEN STOCK ITEM                                   |
                                                                        | 4) DELETE ENTIRE STOCK ITEMS                                   |
                                                                        | 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED      |
                                                                        | 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT)     |
                                                                        | 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM                     |
                                                                        | 8) EXIT                                                        |
                                                                        |________________________________________________________________|

Enter your choice : 7

Enter the STOCK ITEM TO BE MODIFIED : Maggi

WHICH YOU WANT TO MODIFY : 1) PRICE
                           2) PROFIT

Enter your CHOICE : 1
Enter the PRICE OF THE STOCK ELEMENT  : 70
                        The PRICE OF The STOCK ITEM is SUCCESSFULLY MODIFIED


                                                                         ________________________________________________________________
                                                                        |                          START MENU                            |
                                                                        |________________________________________________________________|
                                                                        | 1) INPUT STOCK ITEMS                                           |
                                                                        | 2) DISPLAY STOCK ITEMS                                         |
                                                                        | 3) DELETE A GIVEN STOCK ITEM                                   |
                                                                        | 4) DELETE ENTIRE STOCK ITEMS                                   |
                                                                        | 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED      |
                                                                        | 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT)     |
                                                                        | 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM                     |
                                                                        | 8) EXIT                                                        |
                                                                        |________________________________________________________________|

Enter your choice :
```



```
Enter your choice : 6


The Maximum Profit we get is : 508

The STOCK ITEMS are :

                ________________________________________________________________________________________________________
               | S. No. |                  ITEM NAME                       |  PRICE (PER KGS)  |  PROFIT (IN RS)   |
               |________|__________________________________________________|___________________|___________________|
               |   1)   |                                              pen |        15         |         5         |
               |   2)   |                                  Card_board_sheet |       120         |        40         |
               |   3)   |                                            Duster |        50         |        10         |
               |   4)   |                                              C.D. |        18         |         6         |
               |   5)   |                                      Horse_Millet |        90         |        34         |
               |   6)   |                                             Wheat |        50         |        19         |
               |   7)   |                                              Rice |        65         |        25         |
               |   8)   |                                   Billsberry_Atta |        72         |        16         |
               |   9)   |                                             Maggi |        70         |        14         |
               |  10)   |                             Dairy_Milk_chocolate |        45         |        12         |
               |  11)   |                                             Bingo |       400         |       220         |
               |  12)   |                                             Spoon |        20         |         7         |
               |  13)   |                                           Bonvita |       300         |        50         |
               |  14)   |                                        Phone_case |       180         |        50         |
               |________|__________________________________________________|___________________|___________________|


                                                                         ________________________________________________________________
                                                                        |                          START MENU                            |
                                                                        |________________________________________________________________|
                                                                        | 1) INPUT STOCK ITEMS                                           |
                                                                        | 2) DISPLAY STOCK ITEMS                                         |
                                                                        | 3) DELETE A GIVEN STOCK ITEM                                   |
                                                                        | 4) DELETE ENTIRE STOCK ITEMS                                   |
                                                                        | 5) DELETE STOCK ITEMS, EXCEPT STOCK ITEMS TO BE PURCHASED      |
                                                                        | 6) DISPLAY STOCK ITEMS TO BE PURCHASED(FOR MAXIMUM PROFIT)     |
                                                                        | 7) MODIFY PROFIT/PRICE OF GIVEN STOCK ITEM                     |
                                                                        | 8) EXIT                                                        |
                                                                        |________________________________________________________________|

Enter your choice : 8


                                                                                     | THANK YOU |
                                                                                     |___________|

Process returned 2 (0x2)   execution time : 578.754 s
Press any key to continue.
```

CODE FOR COMPARISON OF DIFFERENT APPROACHES

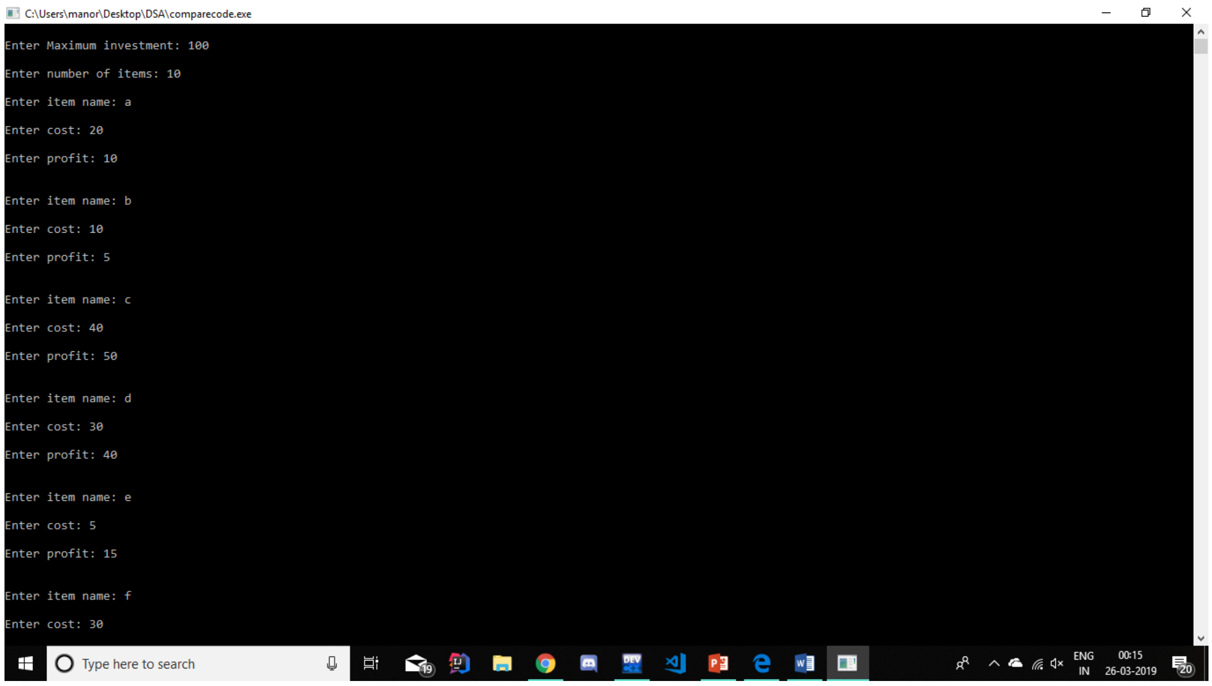
```
#include<iostream>

#include<cstdlib>

#include<cstring>

#include<fstream>

#include<iomanip>

using namespace std;

struct items

{ char name[30];

  int price;

  int profit;

  float density;

}itm[50];


void greedy(items itm[], int num, int W);


int max(int a, int b) { return (a > b)? a : b; }


void dynamicProgramming(int W, items itm[], int n)

{

    int i, w;

    int K[n + 1][W + 1];

    for (i = 0; i <= n; i++) {

        for (w = 0; w <= W; w++) {

            if (i == 0 || w == 0)

                K[i][w] = 0;

            else if (itm[i - 1].price <= w)
```

```
              K[i][w] = max(itm[i-1].profit +  K[i - 1][w - itm[i - 1].price], K[i - 1][w]);
            else
              K[i][w] = K[i - 1][w];
        }
    }
    int res = K[n][W];
    cout<<"\nMaximum Profit= "<< res<<endl;
    w = W;
    for (i = n; i > 0 && res > 0; i--)
           {
        if (res == K[i - 1][w])
            continue;
        else {
            res = res - itm[i-1].profit;
            w = w - itm[i - 1].price;
        }
    }
}
void bruteforce(items itm[],int c,int n)
{
        int i,j,isSelected=1,maxWt,maxProfit;
        int selected[n];
        int iter=2^n;
        int result=0;
        int temp[n];
        for(i =0;i<n;i++)
                temp[i]=0;
        for(i=0;i<iter;i++) {
                isSelected=1;
                maxProfit=0;
                maxWt=0;
        for(j=0;j<n;j++){
                        if(temp[j]==1){
                                maxWt+=itm[j].price;
                                maxProfit+= itm[j].profit;
                        }
                }
                if( maxWt <= c && maxProfit>result){

                        for(j=0;j<n;j++)
                                selected[j]=temp[j];
                        result=maxProfit;
                }
                for(j=0;j<n;j++){
                        temp[j]=temp[j]+isSelected;
                        isSelected = temp[j]/2;
                        temp[j]=temp[j]%2;
                }
        }
        cout<<"\nMaximum Profit="<< result<<endl;
}

void greedy( items itm[], int num, int W)
{
 int i,j, wt,var[num];
 float value;
 float totalWeight = 0, totalBenefit = 0;
items temp;
 for(i = 0; i < num; i++)
 {
   itm[i].density = float(itm[i].profit) / itm[i].price;
   var[i]=0;
 }
 for(i = 1; i < num; i++) {
   for(j = 0; j < num - i; j++) {
     if(itm[j+1].density > itm[j].density) {
       temp = itm[j+1];
       itm[j+1] = itm[j];
       itm[j] = temp;
     }
   }
 }
 int v=W;
 for(i=0;i<num && itm[i].price<=v;i++)
 {
             var[i]=1;
             v=v-itm[i].price;
 }
 for(i=0;i<num;i++)
 {
               totalWeight=totalWeight+(itm[i].price*var[i]);
```

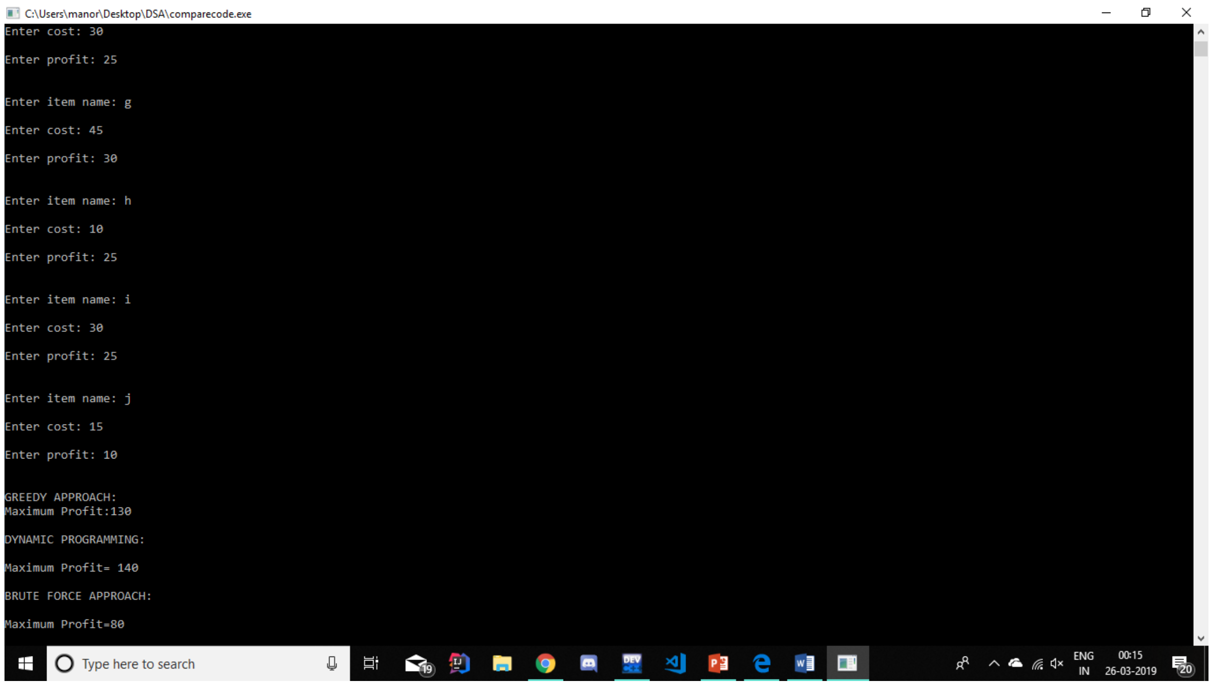
```
              totalBenefit=totalBenefit+(itm[i].profit*var[i]);
  }
   cout<<"Maximum Profit:"<< totalBenefit<<endl;
}

int main(void)
{
  int i, j,W,num,choice;
  cout<<"\nEnter Maximum investment: ";
  cin>>W;
  char ch;
 cout<<"\nEnter number of items: ";
 cin>>num;
  for(i=0;i<num;i++)
  {
                        cout<<"\nEnter item name: ";
                        cin>>itm[i].name;
                        cout<<"\nEnter cost: ";
                        cin>>itm[i].price;
                        cout<<"\nEnter profit: ";
                        cin>>itm[i].profit;
                        cout<<"\n";

            }
            cout<<"\nGREEDY APPROACH:\n";
            greedy(itm,num,W);
    cout<<"\nDYNAMIC PROGRAMMING:\n";
```

```
            dynamicProgramming(W,itm,num);

            cout<<"\nBRUTE FORCE APPROACH:\n";

            bruteforce(itm,W,num);


   return 0;

 }
```

TESTING AND SAMPLE SCREENSHOTS OF OUTPUT:

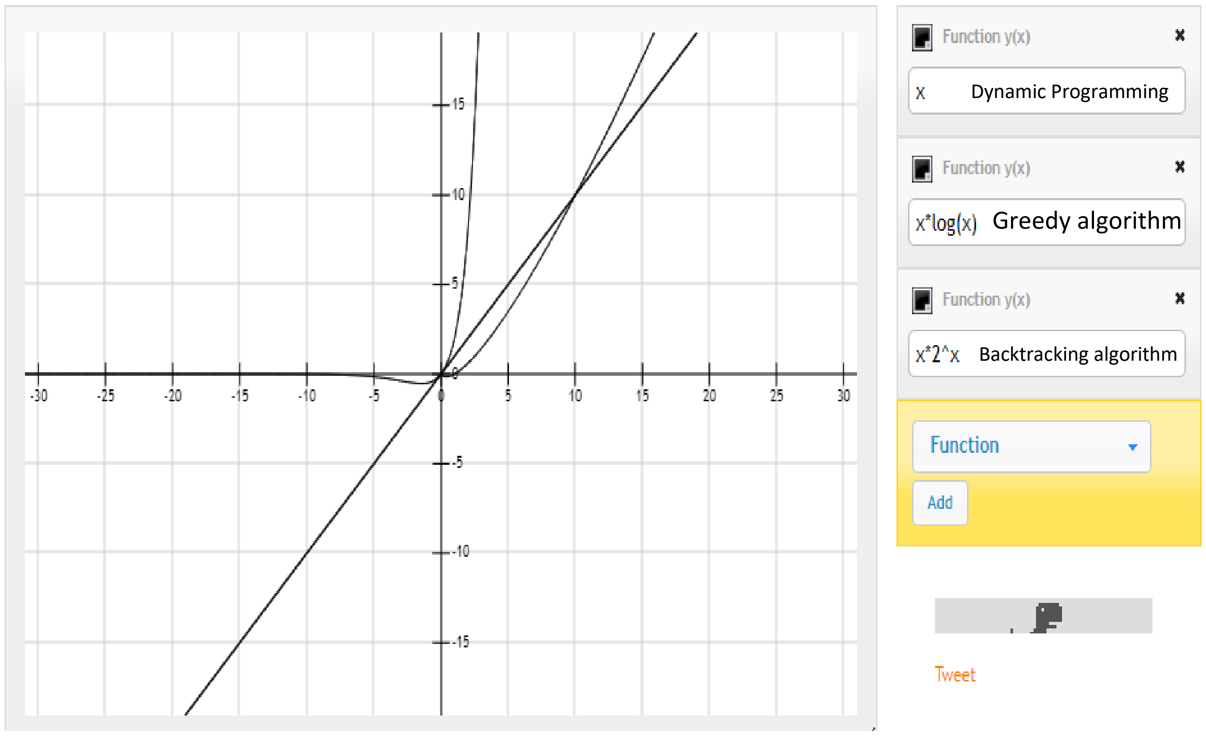


```
Enter Maximum investment: 100
Enter number of items: 10
Enter item name: a
Enter cost: 20
Enter profit: 10

Enter item name: b
Enter cost: 10
Enter profit: 5

Enter item name: c
Enter cost: 40
Enter profit: 50

Enter item name: d
Enter cost: 30
Enter profit: 40

Enter item name: e
Enter cost: 5
Enter profit: 15

Enter item name: f
Enter cost: 30
```



```
Enter cost: 30
Enter profit: 25

Enter item name: g
Enter cost: 45
Enter profit: 30

Enter item name: h
Enter cost: 10
Enter profit: 25

Enter item name: i
Enter cost: 30
Enter profit: 25

Enter item name: j
Enter cost: 15
Enter profit: 10

GREEDY APPROACH:
Maximum Profit:130
DYNAMIC PROGRAMMING:
Maximum Profit= 140
BRUTE FORCE APPROACH:
Maximum Profit=80
```

GRAPH COMPARING THE DIFFERENT APPROACHES:



Function y(x)
x Dynamic Programming

Function y(x)
x*log(x) Greedy algorithm

Function y(x)
x*2^x Backtracking algorithm

CONCLUSION:

Clearly, from the given graph and output, we see that dynamic programming is the most efficient programming technique in order to solve the 0/1 Knapsack Problem. From the graph, we infer that the time complexity of dynamic programming for large outputs is least as compared to the rest.

Hence, using dynamic programming in knapsack, we solve the basic problem that a start-up company faces, that is, where to invest with the given initial investment such that we get maximum profit in the end and do not incur any loses.

REFERENCES:

- https://www.geeksforgeeks.org/branch-and-bound-algorithm/
- https://dev.to/vickylai/knapsack-problem-algorithms-for-my-real-life-carry-on-knapsack-33jj
- http://condor.depaul.edu/ichu/csc491/notes/wk8/knapsack.html
- http://www.micsymposium.org/mics_2005/papers/paper102.pdf