



**C++ PROGRAMMING**

(335)

**REGIONAL 2022**

**APPLICATION KNOWLEDGE:**

Natural Language Store Menu Interface (350 points)

***TOTAL POINTS (350 points)***

**Test Time: 90 minutes**

**GENERAL GUIDELINES:**

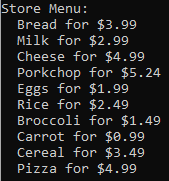
*Failure to adhere to any of the following rules will result in disqualification:*

1. Member must hand in this test booklet and all printouts if any.
2. No equipment, supplies, or materials other than those specified for this event are allowed in the testing area. No previous BPA tests and/or sample tests (handwritten, photocopied, or keyed) are allowed in the testing area.
3. Electronic devices will be monitored according to ACT standards.

**Natural Langue Store Menu Interface**

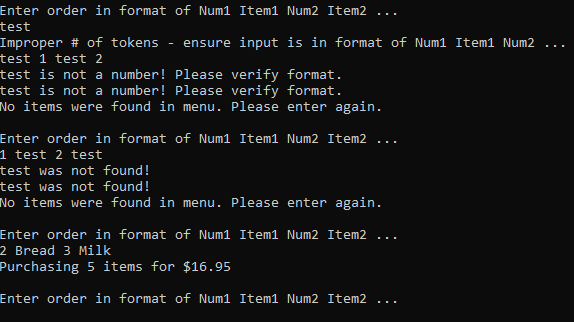
In this exercise, you will create a C++ console application that parses store menu data from a file and provides a natural language interface to be able to purchase items. This includes parsing the menu file for item and cost information, interpreting input to verify format is matched, and then determining the number of items purchased as well as the cost of those items.

On application start, the file will be loaded, and the contents will be presented in the following format:



**Figure 1: Display of store menu contents from loaded file**

Now, the user will be prompted to enter their order in the following fashion as shown in Figure 2. Note that any deviations from the format, be it mismatch of Number/Item tokens must be prompted for re-entry, while a missing item from the store menu shall allow for the rest of the items to be purchased but report that the item was missing. The string “exit” shall be used to exit the application, and until this string is entered the application will continuously prompt for another order.

****

**Figure 2: Display of user input, error messages, and successful order**

Requirements:

1. You must create a C++ console application named CPP\_335\_ContestantNumber, where ContestantNumber is your BPA assigned contestant number (including dashes). For example, CPP\_335\_01\_2345\_6789.
2. Your contestant number must appear as a comment at the top of the main source code file.
3. When application starts, application shall load data from included “menu.txt” file and use that data to display the store menu as shown in Figure 1. Each line in the file is a new item, where a line contains the name of the item and its cost as a double.
   1. If the file was not able to be opened, application shall show an error message indicating that there was an error.
   2. If the data structure of choice is empty after the file is loaded, application shall display an error message that there were no data elements found.
4. After menu is displayed, the user shall be prompted to enter an order in the format specified in Figure 2, where a number precedes the item space-separated and a subsequent item matches the same format.
5. When an order has been placed, the application shall check for an improper number of tokens but continue the application. If “exit” is entered, the application shall exit.
6. Two functions, getSumFromText and getCostFromText must exist to parse the data structure of the menu and the user input to retrieve the number of purchased items as well as the total cost of those items, respectively.
   1. The getCostFromText function shall verify that the items prompted for purchase are indeed in the store’s menu, and if not, an error prompt will be displayed but purchase will still be allowed to proceed on the other items. When calculating cost, remember to take the number of that purchased item and multiply by its cost (2 Milk will cost 2\*$2.99).

Your application will be graded on the following criteria:

**Solution and Project**

The project is present on the flash drive \_\_\_\_ 10 pts

The project is named according to the naming conventions \_\_\_\_ 10 pts

**Program Execution**

Code copied to USB drive and the program runs from USB \_\_\_\_ 10 pts

*If the program does not execute, then the remaining items in this section receive a score of zero.*

Application displays error message if cannot load file \_\_\_\_ 30 pts

Application displays error message if no elements in loaded file \_\_\_\_ 30 pts

Application successfully loads menu from file and stores it \_\_\_\_ 40 pts

Application properly handles error conditions \_\_\_\_ 40 pts

Application displays proper results for purchased items \_\_\_\_ 30 pts

**Source Code Review**

Code is commented at the top, for each function, and as needed \_\_\_\_ 20 pts

Code uses reasonable and consistent variable naming conventions \_\_\_\_ 20 pts

The specified functions are present and are constructed in a logical manner \_\_\_\_ 50 pts

File error handling is performed properly \_\_\_\_ 30 pts

Main function performs only reporting functionality \_\_\_\_ 30 pts

**Total Points**: 350 pts

**Source Code:**

#include <iostream>

#include <string>

#include <sstream>

#include <vector>

#include <fstream>

using namespace std;

void split(string str, vector<string>& tokens);

bool displayMenu(string path, vector<string>& items, vector<double>& costs);

int getSumFromText(string str);

double getCostFromText(vector<string> tokens, vector<string> items, vector<double> costs);

int main()

{

string in = "";

int sum = 0;

double cost = 0.0;

vector<string> items;

vector<double> costs;

if (displayMenu("menu.txt", items, costs))

{

if (items.size() > 0)

{

cout << endl << "Enter order in format of Num1 Item1 Num2 Item2 ..." << endl;

while (getline(cin, in))

{

if (!strcmp(in.c\_str(), "exit")) break;

vector<string> tokens;

split(in, tokens);

if (tokens.size() % 2 != 0)

{

cout << "Improper # of tokens - ensure input is in format of Num1 Item1 Num2 ..." << endl;

continue;

}

sum = getSumFromText(in);

cost = getCostFromText(tokens, items, costs);

if (sum > 0)

{

if (cost > 0.0)

{

cout << "Purchasing " << sum << " items for $" << cost << endl;

}

else

{

cout << "No items were found in menu. Please enter again." << endl;

}

}

else

{

cout << "No quantity was defined in input. Please enter again." << endl;

}

cout << endl << "Enter order in format of Num1 Item1 Num2 Item2 ..." << endl;

}

}

else

{

cout << "No items loaded from menu file." << endl;

}

}

else

{

cout << "Error loading menu file." << endl;

}

}

void split(string str, vector<string>& tokens)

{

string val;

stringstream sstr(str);

while (sstr >> val)

{

tokens.push\_back(val);

}

}

bool displayMenu(string path, vector<string>& items, vector<double>& costs)

{

string line = "";

ifstream file(path);

if (file.is\_open())

{

cout << "Store Menu: " << endl;

while (getline(file, line))

{

vector<string> tokens;

split(line, tokens);

double tok1 = stod(tokens[1]);

cout << " " << tokens[0] << " for $" << tok1 << endl;

items.push\_back(tokens[0]);

costs.push\_back(tok1);

}

return true;

}

else

{

return false;

}

}

int getSumFromText(string str)

{

int num = 0;

int sum = 0;

for (int i = 0; i < (int)str.size(); i++)

{

if (isdigit(str[i])) continue;

else str[i] = ' ';

}

stringstream sstr(str);

while (sstr >> num)

{

sum += num;

}

return sum;

}

double getCostFromText(vector<string> tokens, vector<string> items, vector<double> costs)

{

double sum = 0;

for (int i = 0; i < tokens.size(); i += 2)

{

int num = atoi(tokens[i].c\_str());

if (!tokens[i].empty() && tokens[i].find\_first\_not\_of("0123456789") == string::npos)

{

string item = tokens[i + 1];

bool found = false;

int j = 0;

while (j < items.size() && !found)

{

if (item == items[j])

{

found = true;

}

else

{

j++;

}

}

if (found)

{

sum += costs[j] \* double(num);

}

else

{

cout << item << " was not found!" << endl;

}

}

else

{

cout << tokens[i] << " is not a number! Please verify format." << endl;

}

}

return sum;

}