1 Objectives

This assignment is designed to get you started with using C++ as an implementation tool, giving you practice with arrays, pointers, dynamic memory allocation and deal-location, and with writing classes.

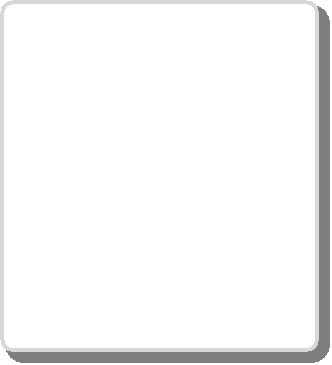
If you are new to the C++ language, you might nd this assignment challenging at rst while learning about pointers, references, dynamic memory, etc. However, you will be happy to know that these notions will become second nature to you very quickly, and that you will hopefully feel pleased about the knowledge you will gain through these assignments.

2 Your Task

Implement a class, named Menu, that models menus used in text-based menu-driven programs, where the user is rst presented with a list of options to choose from and is then prompted to enter a value corresponding to an option, very similar to menu-based voice interfaces.

The string representation of the Menu objects of interest in this assignment look like the textual pattern shown at right.

A Menu object includes of six items:

Opening message Top prompt

(1) option one (2) option two (3) option three (..) ...

(n) option n Closing message

?-> Bottom prompt

the string literal \?->",

four Text items representing the top/bottom prompts and the opening/closing messages, and

a dynamic list of Text items representing the menu’s list of options,

where Text is a class that represents character strings.

The string literal item \?->" provides the minimal string representation for a Menu object regardless of the presence or absence of the other ve items.

Assuming that, for most of you, this might be a rst encounter with programming in C++, this assignment will provide you with detailed UML class diagrams of classes Text and Menu as well as presenting sample program runs.

3 Class Text

Represents a character string and provides a few operations including those involved in the following code segment:

1 int demoText() 2 {

3 Text t1("Welcome to C++"); // conversion constructor 4 Text t2; // defalt constructor

5 Text t3{ t1 }; // copy constructor

6 cout << "t1: " << t1 << endl; // operator<< overload 7 cout << "t2: " << t2 << endl;

8 cout << "t3: " << t3 << endl;

9

10 t2.set(" Programming"); // set t2’s text to " Programming" 11 cout << "t2: " << t2 << endl;

12

13 t3.set(t1); // set t3’s text to t1’s text 14 cout << "t3: " << t3 << endl;

15

16 t1.append(" Programming"); // append the c-string " Programming" to t1’s text 17 cout << "t1: " << t1 << endl;

18

19 t3.append(t2); // append t2’s text to t3’s text 20 cout << "t3: " << t3 << endl;

21

22 return 0; 23 }

output

1 t1: Welcome to C++ 2 t2:

3 t3: Welcome to C++ 4 t2: Programming

5 t3: Welcome to C++ 6 t1: Welcome to C++ 7 t3: Welcome to C++

Programming Programming

For this part, you will implement the following UML class diagram for class Text:

|  |  |  |
| --- | --- | --- |
| Text | | The class Text  points to where the text in this object is stored Destructor. Releases storage in use by this object. |
| { \*text: char | |
| + virtual Text() : | |
| + Text() : | | Default Constructor. same as Text("") |
| + Text( t : const Text & ) : | | Copy Constructor. |
| + | Text( t : const char \* ) : | Conversion Constructor |
| + set( t : char \* ) : void | | Sets this object’s text to \*t |
| + set( t : const Text &) : void | | Sets this object’s text to that of t |
| + operator=( t : const Text & ) : Text & | | Assignment operator= overload. |
| + append( t : const char \* ) : void | | appends \*t’s text to this object’s text. |
| + append( t : const Text & ) : void | | appends t’s text to this object’s text. |
| + length() const : int | | Return the length of this object’s text. |
| + isEmpty() const : bool | | Determines whether this object’s text is empty. |

3.1 Specic Requirements

Your implementation

must store the character strings using raw C-arrays, which must be allocated dynamically using new, and deallocated using delete.

must overload the insertion operator<<.

4 Class Menu

For simplicity, this class provides only a default constructor that creates an default menu. The string representation of a default menu is the string literal \?->".

Class Menu overloads the operator<< to display the string representation of a menu. For example:

1 Menu menu; // an empty menu 2 cout << menu << endl;

Output

1 ?->

Obviously, it would be pointless to create a menu object and just display it. The primary purpose of a menu object is to interact with the user: it displays the menu and expect the user to enter an input value.

The range of input values acceptable by a menu object depends on the number of options in a menu’s option list. If the menu’s option list is not empty, then the user must enter an integer in the range 1 through the highest option number in the list.

However, If the menu’s option list is empty, then the user can enter any integer:

3 int choice = menu.read\_option\_number();

4 cout << "you entered: " << choice << endl;

Output

2 ?-> 1234

3 you entered: 1234

The read option number() method displays the menu (the same as operator<<) and then reads and returns the integer input. Moreover, if the menu’s option list is not employ, read option number() validates the input values, repeatedly rejecting all out-of range input numbers until the user enters a valid option number.

Both read option number() and operator<< delegate their common task of display-ing the menu to another member function named toString() that returns a Text representation of the menu. Thus, the cout statements in the following code segment each display the same output.

cout << menu << endl; // operator<< calls toString() internally on menu

choice = menu.read\_option\_number(); // read\_option\_number() calls toString() internally cout << menu.toString() << endl; // menu calls toString() directly

Text t = menu.toString(); // injects menu’s string representaion into t, a Text object cout << t << endl; // same display as all of the above

Let’s add an option to our menu and then print it:

5 menu.push\_back("Pepsi"); 6 cout << menu << endl;

Output

4

5 (1) Pepsi 6

7 ?->

In a menu with a non-empty option list the display of the option list is preceded and followed by blank lines for better readability.

Let’s add a couple of more options to our menu:

7 menu.push\_back("Apple juice"); 8 menu.push\_back("Root beer");

9 choice = menu.getOptionNumber();

10 cout << "you entered: " << choice << endl;

Output

8

9 (1) Pepsi

10 (2) Apple juice 11 (3) Root beer

12

13 ?-> -1

14 Invalid choice -1. It must be in the range [1, 3]

15

16

17 (1) Pepsi

18 (2) Apple juice 19 (3) Root beer

20

21 ?-> 5

22 Invalid choice 5. It must be in the range [1, 3]

23

24

25 (1) Pepsi

26 (2) Apple juice 27 (3) Root beer

28

29 ?-> 2

30 you entered: 2

Class Menu provides member function to set the prompts in the menu:

11 menu.set\_top\_prompt("Choose your thirst crusher: "); 12 menu.set\_bottom\_prompt("Enter a drink number: ");

13 cout << menu << endl;

Output

31 Choose your thirst crusher: 32

33 (1) Pepsi

34 (2) Apple juice 35 (3) Root beer

36

37 ?-> Enter a drink number:

The following example shows how to remove the last option and then insert a new option at number 2:

14 menu.pop\_back(); // remove the last option

15 menu.insert(1, "Iced tea with lemon"); // this will be option 2 16 choice = menu.read\_option\_number();

17 cout << "you entered: " << choice << endl;

Output

38 Choose your thirst crusher: 39

40 (1) Pepsi

41 (2) Iced tea with lemon 42 (3) Apple juice

43

44 ?-> Enter a drink number: 2 45 you entered: 2

To remove any of the options, the class implements a remove(int) member:

18 menu.pop\_back(); // remove the last option

19 menu.remove(0); // remove the first option (index k indexes option k+1) 20 cout << menu << endl;

Output

46 Choose your thirst crusher: 47

48 (1) Iced tea with lemon 49

50 ?-> Enter a drink number:

The following code segment adds opening and closing messages to the menu:

21 menu.set\_top\_message("Quench your thirst with our fine drinks"); 22 menu.set\_bottom\_message("Time to obey your thirst!");

23 cout << menu << endl;

Output

51 Quench your thirst 52 Choose your thirst

53

with our fine drinks crusher:

54 (1) Iced tea with lemon 55

56 Time to obey your 57 ?-> Enter a drink

thirst! number:

The following code segment removes the only remaining option, leaving this menu with an empty option list:

24 menu.pop\_back();

25 cout << menu << endl;

Output

58 Quench your thirst with our fine drinks 59 Choose your thirst crusher:

60 Time to obey your thirst! 61 ?-> Enter a drink number:

Here is our nal example:

26 menu.set\_top\_message("Who Says You Can’t Buy Happiness?"); 27 menu.clear\_bottom\_message();

28 menu.set\_top\_prompt("Just Consider Our Seriously Delicious Ice Cream Flavors for Summer "); 29 menu.set\_bottom\_prompt("Enter the number of your Happiness Flavor: ");

30 menu.push\_back("Bacon ice cream!");

31 menu.push\_back("Strawberry ice cream"); 32 menu.push\_back("Vanilla ice cream");

33 menu.push\_back("Chocolate chip cookie dough ice cream"); 34 choice = menu.getOptionNumber();

35 cout << "you entered: " << choice << endl;

Output

62 Who Says You Can’t Buy Happiness?

63 Just Consider Our Seriously Delicious Ice Cream Flavors for Summer

64

65 (1) Bacon ice cream!

66 (2) Strawberry ice cream 67 (3) Vanilla ice cream

68 (4) Chocolate chip cookie dough ice cream

69

70 ?-> Enter the number of your Happiness Flavor: 3 71 you entered: 3

A UML class diagram for class Menu is presented as follows:

|  |  |  |
| --- | --- | --- |
| Menu | | The class Menu  The dynamically allocated array storing the options list |
| { option list : Text\* | |
| { count : int | | The number of options in the list |
| { | max list size : int | The maximum number of options supported by current options list |
| { top prompt : Text | | The top prompt |
| { bottom prompt : Text | | The bottom prompt |
| { top text : Text | | The opening message |
| { bottom text : Text | | The closing message |
| { double capacity() : void | | Doubles the current capacity of the options list |
| + | Menu() : | Constructor. Initializes the object and allocates a small options list (See 4.1.) |
| + Menu( otherMenu : const Menu & ) : | | Copy constructor |
| + virtual Menu() : | | Destructor. Releases dynamic storage in use by the options list |
| + Menu& operator =(menu : const Menu & ) : | | Overloads the assignment operator |
| + insert( index : int , option : Text & ) : void | | Inserts option at position index; shifts all options at or past index over to the right by one position. |
| + | push back( pOption : char \* ) : void | Adds supplied option to the end of the option list |
| + push back( option : const Text &) : void | | Adds supplied option to the end of the option list |
| + remove( index : int ) : Text | | Removes an option from the list at given index; shifts all options to the right of index left by one position. Returns the removed option. |
| + size() const : int | | Returns the number of options in the option list. |
| + | capacity() const : int | Returns the maximum number of options that the menu can support until it needs to expand. |
| + pop back() : void | | Removes the last option in the list |
| + get( k : int ) : Text | | Return the k’th option |
| + toString() const : Text | | Returns a Text object storing a string representa-tion of this menu |
| + read option number() : int | | Displays this menu and then reads and returns a valid option number |
| + | set top prompt( t : const Text & ) : void | Sets top prompt to the supplied prompt t |
| + set bottom prompt( t : const Text &):void | | Sets bottom prompt to the supplied prompt t |
| + set top message( m : const Text & ):void | | Sets opening message to m |
| + set bottom message( m :const Text &):void | | Sets closing message to m |
| + clear top prompt() : void | | Removes top prompt |
| + clear bottom prompt() : void | | Removes bottom prompt |
| + clear top message() : void | | Removes opening message |
| + clear bottom message() : void | | Removes closing message |
| + isEmpty() const : bool | | Returns true this menu’s option list is empty |

4.1 Specic Requirements

Your implementation

muststoretheTextobjectsusingrawC-arrays, whichmustbeallocateddynamically using new, and deallocated using delete.

must use the initial capacity of 1 when an object is created by the default constructor. This will speedup testing of your double capacity() method (1 to 2 to 4 to 8, etc.)

must overload the insertion operator<<.

5 General Requirements

Your implementation

may not use the C++ [string](http://www.cplusplus.com/reference/string/string/) class, except for c str() and data(),

may use only the functions strcpy and strlen from the [<cstring>](http://www.cplusplus.com/reference/cstring/) header le,

may use any \String conversion" functions from [<cstdlib>](http://www.cplusplus.com/reference/cstdlib/) and any function from [<cctype>.](http://www.cplusplus.com/reference/cctype/)

may introduce any number of private member functions of your own to facil-itate your tasks.