# 31.1 LAB: Warm up: Hello world

This section has been set as optional by your instructor.



*This zyLab activity prepares a student for a full programming assignment. Warm up exercises are typically simpler and worth fewer points than a full programming assignment, and are well-suited for an in-person scheduled lab meeting or as self-practice.*

For each of the following steps, end the program's output with a newline.

(1) Write a program that outputs the following. (1 pt)

```
Hello world!
```

(2) Update to output the following. (1 pt)

```
Hello world!
How are you?
```

(3) Finally, update to output the following. (1 pt)

```
Hello world!
How are you?
   (I'm fine).
```



LAB ACTIVITY | 31.1.1: LAB: Warm up: Hello world | 0 / 3



main.cpp — Load default template...

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5
6    /* Type your code here. */
```

```
7
8     return 0;
9 }
```



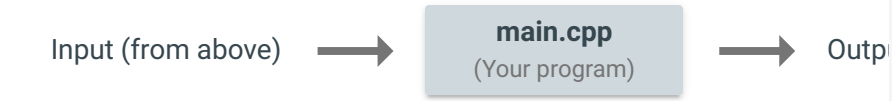
Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program




Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

# 31.2 LAB: Warm up: Basic output with variables



This section has been set as optional by your instructor.

*This zyLab activity prepares a student for a full programming assignment. Warm up exercises are typically simpler and worth fewer points than a full programming assignment, and are well-suited for an in-person scheduled lab meeting or as self-practice.*

A variable like userNum can store a value like an integer. Extend the given program as indicated.

1. Output the user's input. (2 pts)
2. Output the input squared and cubed. *Hint: Compute squared as userNum * userNum.* (2 pts)
3. Get a second user input into userNum2, and output the sum and product. (1 pt)

*Note: This zyLab outputs a newline after each user-input prompt. For convenience in the examples below, the user's input value is shown on the next line, but such values don't actually appear as output when the program runs.*



```
Enter integer:
4
You entered: 4
4 squared is 16
And 4 cubed is 64!!
Enter another integer:
5
4 + 5 is 9
4 * 5 is 20
```

539740.3879454.qx3zqy7

**LAB ACTIVITY** 31.2.1: LAB: Warm up: Basic output with variables 0 / 10

main.cpp Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     int userNum;
6
7     cout << "Enter integer:" << endl;
8     cin  >> userNum;
9
10    /* Type your code here */
11
12    return 0;
13 }
```



Develop mode Submit mode Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first

box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.



Run program

Input (from above) → main.cpp (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 31.3 LAB*: Program: ASCII art

This section has been set as optional by your instructor.

*This zyLab activity is the traditional programming assignment, typically requiring a few hours over a week. The previous sections provide warm up exercises intended to help a student prepare for this programming assignment.*

(1) Output this tree. (2 pts)



```
   *
  ***
 *****
*******
  ***
```

(2) Below the tree (with two blank lines), output this cat. (3 pts)

```
/\   /\
  o o
 =   =
  ---
```

*Hint: A backslash `\` in a string acts as an escape character, such as with a newline `\n`. So, to print an actual backslash, escape that backslash by prepending another backslash. Ex: The following prints a single backslash:* `cout << "\\";`





**LAB ACTIVITY** | 31.3.1: LAB*: Program: ASCII art | 0 / 5

main.cpp — Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5
6     // Draw tree
7     cout << "   *"    << endl;
8     cout << "  ***"   << endl;
9     /* Type your code here. */
10
11    return 0;
12 }
```

**Develop mode** | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.



**Run program** Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```



# 31.4 LAB: Warm up: Variables, input, and casting

This section has been set as optional by your instructor.

(1) Prompt the user to input an integer, a double, a character, and a string, storing each into separate variables. Then, output those four values on a single line separated by a space. (2 pts)

*Note: This zyLab outputs a newline after each user-input prompt. For convenience in the examples below, the user's input value is shown on the next line, but such values don't actually appear as output when the program runs.*

```
Enter integer:
99
Enter double:
3.77
Enter character:
z
Enter string:
Howdy
99 3.77 z Howdy
```



(2) Extend to also output in reverse. (1 pt)

```
Enter integer:
99
Enter double:
3.77
Enter character:
```

```
z
Enter string:
Howdy
99 3.77 z Howdy
Howdy z 3.77 99
```

(3) Extend to cast the double to an integer, and output that integer. (2 pts)



```
Enter integer:
99
Enter double:
3.77
Enter character:
z
Enter string:
Howdy
99 3.77 z Howdy
Howdy z 3.77 99
3.77 cast to an integer is 3
```



**LAB ACTIVITY** 31.4.1: LAB: Warm up: Variables, input, and casting 0 / 5

main.cpp Load default template...

```
#include <iostream>
#include <string>      // Supports use of "string" data type
using namespace std;

int main() {
   int    userInt;
   double userDouble;
   // FIXME: Define char and string variables

   cout << "Enter integer:" << endl;
   cin  >> userInt;

   // FIXME (1): Finish reading other items into variables, then output the

   // FIXME (2): Output the four values in reverse
```



Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first

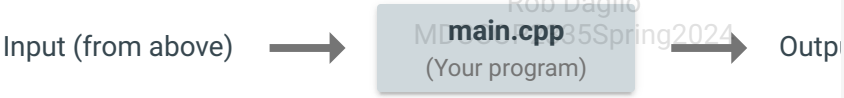
box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.



Run program




Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 31.5 LAB: Warm up: Text message abbreviation decoder

This section has been set as optional by your instructor.

(1) Write a program that prompts a user to enter an abbreviation. If the user's input string matches a known text message abbreviation, output the unabbreviated form, else output: Unknown. Support two abbreviations: LOL -- laughing out loud, and IDK -- I don't know. (4 pts)

If the input is:



```
LOL
```

the output is:

```
Input an abbreviation:
laughing out loud
```

If the input is:

```
TTYL
```

the output is:

```
Input an abbreviation:
Unknown
```



Note: The strings "LOL" and "TTYL" are not present in the display of the zyBook environment because both strings are inputs and therefore not part of the program output.

(2) Expand to also decode these abbreviations. (3 pts)

- BFF -- best friends forever
- IMHO -- in my humble opinion
- TMI -- too much information



**LAB ACTIVITY** 31.5.1: LAB: Warm up: Text message abbreviation decoder 0 / 7

main.cpp Load default template...

```
#include <iostream>
#include <string> // Note: This library is needed to use the string type
using namespace std;

int main() {

   /* Type your code here. */

   return 0;
}
```



Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Outp



Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working on this zyLab.
```

# 31.6 LAB*: Program: Text message decoder

This section has been set as optional by your instructor.

(1) Use getline() to get a line of user input into a string. Output the line. (3 pts)

Ex:

```
Enter text:
IDK if I'll go. It's my BFF's birthday.
You entered: IDK if I'll go. It's my BFF's birthday.
```

(2) Search the string (using find()) for common abbreviations and print a list of each found abbreviation along with its decoded meaning. (3 pts)



Ex:

```
Enter text:
IDK if I'll go. It's my BFF's birthday.
You entered: IDK if I'll go. It's my BFF's birthday.
```

```
BFF: best friend forever
IDK: I don't know
```

Support these abbreviations:

- BFF -- best friend forever
- IDK -- I don't know
- JK -- just kidding
- TMI -- too much information
- TTYL -- talk to you later





**LAB ACTIVITY** | 31.6.1: LAB*: Program: Text message decoder | 0 / 6

main.cpp — Load default template...

```
1  #include <iostream>
2  // FIXME include the string library
3  using namespace std;
4
5  int main() {
6
7     /* Type your code here. */
8
9     return 0;
10 }
```

**Develop mode** | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.



**Run program** Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```



# 31.7 LAB*: Program: Text message expander

This section has been set as optional by your instructor.

(1) Get a line of text from the user. Output that line. (1 pt)

Ex:

```
Enter text:
IDK how that happened. TTYL.
You entered: IDK how that happened. TTYL.
```

(2) Output the line again, this time expanding common text message abbreviations. (5 pts)

Ex:

```
Enter text:
IDK how that happened. TTYL.
You entered: IDK how that happened. TTYL.
Expanded: I don't know how that happened. talk to you later.
```



Support these abbreviations:

- BFF -- best friend forever
- IDK -- I don't know
- JK -- just kidding
- TMI -- too much information
- TTYL -- talk to you later

Note: If an abbreviation appears twice, only expand its first instance.

539740.3879454.qx3zqy7

**LAB ACTIVITY** | 31.7.1: LAB*: Program: Text message expander | 0 / 6

main.cpp

Load default template...



```
#include <iostream>
// FIXME include string library
using namespace std;

int main() {

   /* Type your code here. */

   return 0;
}
```

**Develop mode** | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

**Run program**

Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here



Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 31.8 LAB: Warm up: Drawing a right triangle



This section has been set as optional by your instructor.

This program will output a right triangle based on user specified height triangleHeight and symbol triangleChar.

(1) The given program outputs a fixed-height triangle using a * character. Modify the given program to output a right triangle that instead uses the user-specified triangleChar character. (1 pt)

(2) Modify the program to use a nested loop to output a right triangle of height triangleHeight. The first line will have one user-specified character, such as % or *. Each subsequent line will have one additional user-specified character until the number in the triangle's base reaches triangleHeight. Output a space after each user-specified character, including after the line's last user-specified character. (2 pts)

Example output for triangleChar = % and triangleHeight = 5:

```
Enter a character:
%
Enter triangle height:
5

%
% %
% % %
% % % %
% % % % %
```





LAB ACTIVITY | 31.8.1: LAB: Warm up: Drawing a right triangle | 0 / 3

main.cpp Load default template...

```
#include <iostream>
using namespace std;

int main() {
   char triangleChar;
   int triangleHeight;

   cout << "Enter a character:" << endl;
   cin >> triangleChar;

   cout << "Enter triangle height:" << endl;
   cin >> triangleHeight;
   cout << endl;

   cout << "*" << " " << endl;
```



Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.



# 31.9 LAB: Warm up: People's weights (Vectors)

This section has been set as optional by your instructor.

Output each floating-point value with two digits after the decimal point, which can be achieved by executing
`cout << fixed << setprecision(2);` once before all other cout statements.

(1) Prompt the user to enter five numbers, being five people's weights. Store the numbers in a vector of doubles. Output the vector's numbers on one line, each number followed by one space. (2 pts)



Ex:

```
Enter weight 1:
236.0
Enter weight 2:
89.5
Enter weight 3:
142.0
Enter weight 4:
166.3
Enter weight 5:
93.0
You entered: 236.00 89.50 142.00 166.30 93.00
```

(2) Also output the total weight, by summing the vector's elements. (1 pt)

(3) Also output the average of the vector's elements. (1 pt)

(4) Also output the max vector element. (2 pts)

Ex:

```
Enter weight 1:
236.0
Enter weight 2:
89.5
Enter weight 3:
142.0
Enter weight 4:
166.3
Enter weight 5:
93.0
You entered: 236.00 89.50 142.00 166.30 93.00

Total weight: 726.80
```

```
Average weight: 145.36
Max weight: 236.00
```

539740.3879454.qx3zqy7

**LAB ACTIVITY** 31.9.1: LAB: Warm up: People's weights (Vectors) 0 / 6



main.cpp

Load default template...

```
#include <iostream>
#include <iomanip>                // For setprecision
// FIXME include vector library
using namespace std;

int main() {

   /* Type your code here. */

   return 0;
}
```

**Develop mode** | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

**Run program**

Input (from above) → **main.cpp** (Your program) → Outp



Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 31.10 LAB*: Program: Soccer team roster (Vectors)



This section has been set as optional by your instructor.

This program will store roster and rating information for a soccer team. Coaches rate players during tryouts to ensure a balanced team.

(1) Prompt the user to input five pairs of numbers: A player's jersey number (0 - 99) and the player's rating (1 - 9). Store the jersey numbers in one int vector and the ratings in another int vector. Output these vectors (i.e., output the roster). (3 pts)

Ex:

```
Enter player 1's jersey number:
84
Enter player 1's rating:
7

Enter player 2's jersey number:
23
Enter player 2's rating:
4

Enter player 3's jersey number:
4
Enter player 3's rating:
5

Enter player 4's jersey number:
30
Enter player 4's rating:
2

Enter player 5's jersey number:
```

```
66
Enter player 5's rating:
9

ROSTER
Player 1 -- Jersey number: 84, Rating: 7
Player 2 -- Jersey number: 23, Rating: 4
...
```



(2) Implement a menu of options for a user to modify the roster. Each option is represented by a single character. The program initially outputs the menu, and outputs the menu after a user chooses an option. The program ends when the user chooses the option to Quit. For this step, the other options do nothing. (2 pts)

Ex:

```
MENU
a - Add player
d - Remove player
u - Update player rating
r - Output players above a rating
o - Output roster
q - Quit

Choose an option:
```

(3) Implement the "Output roster" menu option. (1 pt)

Ex:

```
ROSTER
Player 1 -- Jersey number: 84, Rating: 7
Player 2 -- Jersey number: 23, Rating: 4
...
```

(4) Implement the "Add player" menu option. Prompt the user for a new player's jersey number and rating. Append the values to the two vectors. (1 pt)



Ex:

```
Enter a new player's jersey number:
49
Enter the player's rating:
8
```

(5) Implement the "Delete player" menu option. Prompt the user for a player's jersey number. Remove the player from the roster (delete the jersey number and rating). (2 pts)

Ex:

```
Enter a jersey number:
4
```



(6) Implement the "Update player rating" menu option. Prompt the user for a player's jersey number. Prompt again for a new rating for the player, and then change that player's rating. (1 pt)

Ex:

```
Enter a jersey number:
23
Enter a new rating for player:
6
```

(7) Implement the "Output players above a rating" menu option. Prompt the user for a rating. Print the jersey number and rating for all players with ratings above the entered value. (2 pts)

Ex:

```
Enter a rating:
5

ABOVE 5
Player 1 -- Jersey number: 84, Rating: 7
...
```



**LAB ACTIVITY** | 31.10.1: LAB*: Program: Soccer team roster (Vectors) | 0 / 12

main.cpp — Load default template...



```
#include <iostream>
// FIXME include vector library
using namespace std;

int main() {

   /* Type your code here. */

   return 0;
```

```
10 }
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.



Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

# 31.11 LAB: Warm up: Text analyzer & modifier

This section has been set as optional by your instructor.



(1) Prompt the user to enter a string of their choosing. Output the string. (1 pt)

Ex:

```
Enter a sentence or phrase:
The only thing we have to fear is fear itself.
```

```
You entered: The only thing we have to fear is fear itself.
```

(2) Complete the GetNumOfCharacters() function, which returns the number of characters in the user's string. *We encourage you to use a for loop in this function.* (2 pts)

(3) In main(), call the GetNumOfCharacters() function and then output the returned result. (1 pt)

(4) Implement the OutputWithoutWhitespace() function. OutputWithoutWhitespace() outputs the string's characters except for whitespace (spaces, tabs). Note: A tab is '\t'. Call the OutputWithoutWhitespace() function in main(). (2 pts)



Ex:

```
Enter a sentence or phrase:
The only thing we have to fear is fear itself.

You entered: The only thing we have to fear is fear itself.

Number of characters: 46
String with no whitespace: Theonlythingwehavetofearisfearitself.
```



**LAB ACTIVITY** 31.11.1: LAB: Warm up: Text analyzer & modifier 0 / 6

main.cpp Load default template...

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  //Returns the number of characters in usrStr
6  int GetNumOfCharacters(const string usrStr) {
7
8     /* Type your code here. */
9
10 }
11
12 int main() {
13
14    /* Type your code here. */
15
```



Develop mode Submit mode Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first

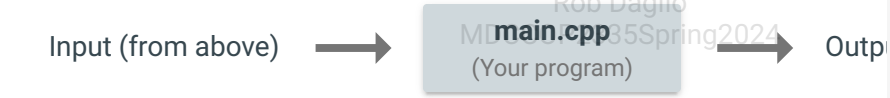
box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.



Run program




Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 31.12 LAB*: Program: Authoring assistant

This section has been set as optional by your instructor.

(1) Prompt the user to enter a string of their choosing. Store the text in a string. Output the string. (1 pt)

Ex:

```
Enter a sample text:
We'll continue our quest in space.  There will be more shuttle flights
and more shuttle crews and,  yes,  more volunteers, more civilians,
more teachers in space.  Nothing ends here;  our hopes and our journeys
continue!

You entered: We'll continue our quest in space.  There will be more
shuttle flights and more shuttle crews and,  yes,  more volunteers,
```

```
more civilians,  more teachers in space.  Nothing ends here;  our hopes
and our journeys continue!
```

(2) Implement the PrintMenu() function to print the following command menu. (1 pt)

Ex:



```
MENU
c - Number of non-whitespace characters
w - Number of words
f - Find text
r - Replace all !'s
s - Shorten spaces
q - Quit
```

(3) Implement the ExecuteMenu() function that takes 2 parameters: a character representing the user's choice and the user provided sample text. ExecuteMenu() performs the menu options, according to the user's choice, by calling the appropriate functions described below. (1 pt)

(4) In the main() function, call PrintMenu() and prompt for the user's choice of menu options for analyzing/editing the string. Each option is represented by a single character.

If an invalid character is entered, continue to prompt for a valid choice. When a valid option is entered, execute the option by calling ExecuteMenu(). Then, print the menu, and prompt for a new option. Continue until the user enters 'q'. *Hint: Implement Quit before implementing other options.* (1 pt)

Ex:

```
MENU
c - Number of non-whitespace characters
w - Number of words
f - Find text
r - Replace all !'s
s - Shorten spaces
q - Quit

Choose an option:
```



(5) Implement the GetNumOfNonWSCharacters() function. GetNumOfNonWSCharacters() has a constant string as a parameter and returns the number of characters in the string, excluding all whitespace. Call GetNumOfNonWSCharacters() in the ExecuteMenu() function, and then output the

returned value. (4 pts)

Ex:

```
Enter a sample text:
We'll continue our quest in space.  There will be more shuttle flights
and more shuttle crews and,  yes,  more volunteers, more civilians,
more teachers in space.  Nothing ends here;  our hopes and our journeys
continue!

You entered: We'll continue our quest in space.  There will be more
shuttle flights and more shuttle crews and,  yes,  more volunteers,
more civilians,  more teachers in space.  Nothing ends here;  our hopes
and our journeys continue!

MENU
c - Number of non-whitespace characters
w - Number of words
f - Find text
r - Replace all !'s
s - Shorten spaces
q - Quit

Choose an option:
c
Number of non-whitespace characters: 181
```



(6) Implement the GetNumOfWords() function. GetNumOfWords() has a constant string as a parameter and returns the number of words in the string. *Hint: Words end when a space is reached except for the last word in a sentence.* Call GetNumOfWords() in the ExecuteMenu() function, and then output the returned value. (3 pts)

Ex:

```
Number of words: 35
```



(7) Implement the FindText() function, which has two strings as parameters. The first parameter is the text to be found in the user provided sample text, and the second parameter is the user provided sample text. The function returns the number of instances a word or phrase is found in the string. In the ExecuteMenu() function, prompt the user for a word or phrase to be found. Then call FindText() and output the returned value. Before the prompt, call **cin.ignore()** to allow the user to input a new string.(3 pts)

Ex:

```
Enter a word or phrase to be found:
more
"more" instances: 5
```



(8) Implement the ReplaceExclamation() function. ReplaceExclamation() has a string parameter and updates the string by replacing each '!' character in the string with a '.' character. ReplaceExclamation() DOES NOT output the string. Call ReplaceExclamation() in the ExecuteMenu() function, and then output the edited string. (3 pts)

Ex.

```
Edited text: We'll continue our quest in space.  There will be more
shuttle flights and more shuttle crews and,  yes,  more volunteers,
more civilians,  more teachers in space.  Nothing ends here;  our hopes
and our journeys continue.
```

(9) Implement the ShortenSpace() function. ShortenSpace() has a string parameter and updates the string by replacing all sequences of 2 or more spaces with a single space. ShortenSpace() DOES NOT output the string. Call ShortenSpace() in the ExecuteMenu() function, and then output the edited string. (3 pt)

Ex:

```
Edited text: We'll continue our quest in space. There will be more
shuttle flights and more shuttle crews and, yes, more volunteers, more
civilians, more teachers in space. Nothing ends here; our hopes and our
journeys continue!
```



**LAB ACTIVITY** | 31.12.1: LAB*: Program: Authoring assistant | 0 / 20



main.cpp — Load default template...

```
1 #include <iostream>
2 #include <string>
3 using namespace std;
4
5 /* Define your functions here. */
6
```

```
7  int main() {
8
9     /* Type your code here. */
10
11    return 0;
12 }
```



Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

# 31.13 LAB*: Program: Online shopping cart (Part 1)



This section has been set as optional by your instructor.

(1) Create three files to submit:

- ItemToPurchase.h - Class declaration
- ItemToPurchase.cpp - Class definition
- main.cpp - main() function

Build the ItemToPurchase class with the following specifications:

- Default constructor
- Public class functions (mutators & accessors)
  - SetName() & GetName() (2 pts)
  - SetPrice() & GetPrice() (2 pts)
  - SetQuantity() & GetQuantity() (2 pts)
- Private data members
  - string itemName - Initialized in default constructor to "none"
  - int itemPrice - Initialized in default constructor to 0
  - int itemQuantity - Initialized in default constructor to 0



(2) In main(), prompt the user for two items and create two objects of the ItemToPurchase class. Before prompting for the second item, call **cin.ignore()** to allow the user to input a new string. (2 pts)

Ex:

```
Item 1
Enter the item name:
Chocolate Chips
Enter the item price:
3
Enter the item quantity:
1

Item 2
Enter the item name:
Bottled Water
Enter the item price:
1
Enter the item quantity:
10
```



(3) Add the costs of the two items together and output the total cost. (2 pts)

Ex:

```
TOTAL COST
Chocolate Chips 1 @ $3 = $3
```

```
Bottled Water 10 @ $1 = $10

Total: $13
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 31.13.1: LAB*: Program: Online shopping cart (Part 1) | 0 / 10



Current file: main.cpp Load default template...

```
1  #include <iostream>
2  using namespace std;
3
4  #include "ItemToPurchase.h"
5
6  int main() {
7
8     /* Type your code here */
9
10    return 0;
11 }
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Outp



Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 31.14 LAB*: Program: Online shopping cart (Part 2)



This section has been set as optional by your instructor.

This program extends the earlier "Online shopping cart (Part 1)" program. (Consider first saving your earlier program).

## Step 1: Extend the ItemToPurchase class per the following specifications:

- Parameterized constructor to assign item name, item description, item price, and item quantity (default values of 0). (1 pt)
- Public member functions
  - `SetDescription()` mutator & `GetDescription()` accessor (2 pts)
  - `PrintItemCost()` - Outputs the item name followed by the quantity, price, and subtotal
  - `PrintItemDescription()` - Outputs the item name and description
- Private data members
  - string `itemDescription` - Initialized in default constructor to "none"

Ex. of `PrintItemCost()` output:

```
Bottled Water 10 @ $1 = $10
```

Ex. of `PrintItemDescription()` output:



```
Bottled Water: Deer Park, 12 oz.
```

## Step 2: Build three new files:

- `ShoppingCart.h` - Class declaration
- `ShoppingCart.cpp` - Class definition

- `main.cpp` - `main()` function *(Note: main()'s functionality differs from the previous program.)*

Build the ShoppingCart class with the following specifications.

- Default constructor
- Parameterized constructor which takes the customer name and date as parameters (1 pt)
- Private data members
  - string `customerName` - Initialized in default constructor to "none"
  - string `currentDate` - Initialized in default constructor to "January 1, 2016"
  - vector < ItemToPurchase > `cartItems`
- Public member functions
  - `GetCustomerName()` accessor (1 pt)
  - `GetDate()` accessor (1 pt)
  - `AddItem()`
    - Adds an item to `cartItems` vector. Has parameter ItemToPurchase. Does not return anything.
  - `RemoveItem()`
    - Removes item from `cartItems` vector. Has a string (an item's name) parameter. Does not return anything.
    - If item name cannot be found, output a message: `Item not found in cart. Nothing removed.`
  - `ModifyItem()`
    - Modifies an item's description, price, and/or quantity. Has parameter ItemToPurchase. Does not return anything.
    - If item can be found (by name) in cart, check if parameter has default values for description, price, and quantity. If not, modify item in cart.
    - If item cannot be found (by name) in cart, output a message: `Item not found in cart. Nothing modified.`
  - `GetNumItemsInCart()` (2 pts)
    - Returns quantity of all items in cart. Has no parameters.
  - `GetCostOfCart()` (2 pts)
    - Determines and returns the total cost of items in cart. Has no parameters.
  - `PrintTotal()`
    - Outputs total of objects in cart.
    - If cart is empty, output a message: `SHOPPING CART IS EMPTY`
  - `PrintDescriptions()`
    - Outputs each item's description.
    - If cart is empty, output a message: `SHOPPING CART IS EMPTY`



Ex. of `PrintTotal()` output:

```
John Doe's Shopping Cart - February 1, 2016
Number of Items: 8
```

```
Nike Romaleos 2 @ $189 = $378
Chocolate Chips 5 @ $3 = $15
Powerbeats 2 Headphones 1 @ $128 = $128

Total: $521
```

Ex. of `PrintDescriptions()` output:



```
John Doe's Shopping Cart - February 1, 2016

Item Descriptions
Nike Romaleos: Volt color, Weightlifting shoes
Chocolate Chips: Semi-sweet
Powerbeats 2 Headphones: Bluetooth headphones
```

# Step 3: In main(), prompt the user for a customer's name and today's date. Output the name and date. Create an object of type ShoppingCart. (1 pt)

Ex:

```
Enter customer's name:
John Doe
Enter today's date:
February 1, 2016

Customer name: John Doe
Today's date: February 1, 2016
```

# Step 4: Implement the PrintMenu() function



- `PrintMenu()`
  - Prints the following menu of options to manipulate the shopping cart. (1 pt)

Ex:

```
MENU
a - Add item to cart
```

```
d - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit
```

- `ExecuteMenu()`
  - Takes 2 parameters: a character representing the user's choice and a shopping cart. Performs the menu options described below in step 5, according to the user's choice. (1 pt)



# Step 5: Implement the menu options

**Step 5a:** In `main()`, call `PrintMenu()` and prompt for the user's choice of menu options. Each option is represented by a single character.

If an invalid character is entered, continue to prompt for a valid choice. When a valid option is entered, execute the option by calling `ExecuteMenu()`. Then, print the menu and prompt for a new option. Continue until the user enters 'q'. (1 pt)

*Hint: Implement Quit before implementing other options.*

Ex:

```
a - Add item to cart
d - Remove item from cart
c - Change item quantity
i - Output items' descriptions
o - Output shopping cart
q - Quit

Choose an option:
```

**Step 5b:** Implement "Output shopping cart" menu option in `ExecuteMenu()`. (3 pts)

Ex:



```
OUTPUT SHOPPING CART
John Doe's Shopping Cart - February 1, 2016
Number of Items: 8

Nike Romaleos 2 @ $189 = $378
Chocolate Chips 5 @ $3 = $15
Powerbeats 2 Headphones 1 @ $128 = $128
```

```
Total: $521
```

**Step 5c:** Implement "Output items' descriptions" menu option in `ExecuteMenu()`. (2 pts)

Ex:

```
OUTPUT ITEMS' DESCRIPTIONS
John Doe's Shopping Cart - February 1, 2016

Item Descriptions
Nike Romaleos: Volt color, Weightlifting shoes
Chocolate Chips: Semi-sweet
Powerbeats 2 Headphones: Bluetooth headphones
```



**Step 5d:** Implement "Add item to cart" menu option in `ExecuteMenu()`. (3 pts)

Ex:

```
ADD ITEM TO CART
Enter the item name:
Nike Romaleos
Enter the item description:
Volt color, Weightlifting shoes
Enter the item price:
189
Enter the item quantity:
2
```

**Step 5e:** Implement "Remove item from cart" menu option in `ExecuteMenu()`. (4 pts)

Ex:

```
REMOVE ITEM FROM CART
Enter name of item to remove:
Chocolate Chips
```

**Step 5f:** Implement "Change item quantity" menu option in `ExecuteMenu()`. (5 pts)

*Hint: Make new ItemToPurchase object and use ItemToPurchase modifiers before using ModifyItem() method.*

Ex:

```
CHANGE ITEM QUANTITY
Enter the item name:
Nike Romaleos
```

```
Enter the new quantity:
3
```

539740.3879454.qx3zqy7

**LAB ACTIVITY** | 31.14.1: LAB*: Program: Online shopping cart (Part 2) | 0 / 31



Current file: main.cpp ▾ Load default template...

```
1  #include <iostream>
2  #include <iomanip>
3  using namespace std;
4
5  #include "ShoppingCart.h"
6
7  void PrintMenu() {
8     /* Type your code here */
9
10 }
11
12 void ExecuteMenu(char option, ShoppingCart& theCart) {
13    /* Type your code here */
14
15 }
```

**Develop mode** | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

**Run program**

Input (from above) → **main.cpp** (Your program) → Outp



Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```

# 31.15 LAB*: Program: Playlist



This section has been set as optional by your instructor.

*You will be building a linked list. Make sure to keep track of both the head and tail nodes.*

## Step 1: Create three files to submit and build the PlaylistNode class.

- `PlaylistNode.h` - Class declaration
- `PlaylistNode.cpp` - Class definition
- `main.cpp` - main() function

Build the PlaylistNode class per the following specifications. *Note: Some functions can initially be function stubs (empty functions), to be completed in later steps.*

- Private data members
  - `string uniqueID` - Initialized to "none" in default constructor
  - `string songName` - Initialized to "none" in default constructor
  - `string artistName` - Initialized to "none" in default constructor
  - `int songLength` - Initialized to 0 in default constructor
  - `PlaylistNode* nextNodePtr` - Initialized to 0 in default constructor
- Default constructor (1 pt)
- Parameterized constructor (1 pt)
- Public member functions
  - `GetID()` - Accessor
  - `GetSongName()` - Accessor
  - `GetArtistName()` - Accessor
  - `GetSongLength()` - Accessor
  - `GetNext()` - Accessor
  - `InsertAfter(PlaylistNode* nodePtr)` - Mutator (1 pt)
  - `SetNext(PlaylistNode* nodePtr)` - Mutator (1 pt)
  - `PrintPlaylistNode()` - Outputs uniqueID, songname, artistName, and songLength based on the format example below.

Ex. of PrintPlaylistNode output:

```
Unique ID: S123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237
```



## Step 2: In main(), prompt the user for the title of the playlist. (1 pt)

Ex:

```
Enter playlist's title:
JAMZ
```

## Step 3: Implement the PrintMenu() function. (1 pt)

PrintMenu() takes the playlist title as a parameter and outputs a menu of options to manipulate the playlist.

Ex:

```
JAMZ PLAYLIST MENU
a - Add song
d - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit
```



## Step 4: Implement the ExecuteMenu() function. (1 pt)

ExecuteMenu() takes 3 parameters: a character representing the user's choice, a playlist title, and the pointer to the head node of a playlist. ExecuteMenu() performs the menu options (described below) according to the user's choice and returns the pointer to the head node of the playlist.

# Step 5: In main(), call PrintMenu() and prompt for the user's choice of menu options. (1 pt)

Each option is represented by a single character. If an invalid character is entered, continue to prompt for a valid choice. When a valid option is entered, execute the option by calling ExecuteMenu(). Then, print the menu, and prompt for a new option. Continue until the user enters 'q'. *Hint: Implement Quit before implementing other options.*



Ex:

```
JAMZ PLAYLIST MENU
a - Add song
d - Remove song
c - Change position of song
s - Output songs by specific artist
t - Output total time of playlist (in seconds)
o - Output full playlist
q - Quit

Choose an option:
```

# Step 6: Implement "Output full playlist" menu option in ExecuteMenu(). (3 pts)

If the list is empty, output: `Playlist is empty`

Ex:

```
JAMZ - OUTPUT FULL PLAYLIST
1.
Unique ID: SD123
Song Name: Peg
Artist Name: Steely Dan
Song Length (in seconds): 237

2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391
```

```
3.
Unique ID: J345
Song Name: Canned Heat
Artist Name: Jamiroquai
Song Length (in seconds): 330

4.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197

5.
Unique ID: SD567
Song Name: I Got The News
Artist Name: Steely Dan
Song Length (in seconds): 306
```



# Step 7: Implement the "Add song" menu option in ExecuteMenu(). (2 pts)

New additions are added to the end of the list.

Ex:

```
ADD SONG
Enter song's unique ID:
SD123
Enter song's name:
Peg
Enter artist's name:
Steely Dan
Enter song's length (in seconds):
237
```



# Step 8: Implement the "Remove song" menu option in ExecuteMenu(). (4 pts)

Prompt the user for the unique ID of the song to be removed.

Ex:

```
REMOVE SONG
Enter song's unique ID:
JJ234
"All For You" removed.
```



# Step 9: Implement the "Change position of song" menu option in ExecuteMenu().

Prompt the user for the current position of the song and the desired new position. Valid new positions are 1 - *n* (the number of nodes). If the user enters a new position that is less than 1, move the node to the position 1 (immediately after the head). If the user enters a new position greater than *n*, move the node to position *n* (the tail). 6 cases will be tested:

- Moving the head node (1 pt)
- Moving the tail node (1 pt)
- Moving a node to the head (1 pt)
- Moving a node to the tail (1 pt)
- Moving a node up the list (1 pt)
- Moving a node down the list (1 pt)

Ex:

```
CHANGE POSITION OF SONG
Enter song's current position:
3
Enter new position for song:
2
"Canned Heat" moved to position 2
```

# Step 10: Implement the "Output songs by specific artist" menu option in ExecuteMenu(). (2 pt)



Prompt the user for the artist's name, and output the node's information, starting with the node's current position.

Ex:

```
OUTPUT SONGS BY SPECIFIC ARTIST
Enter artist's name:
Janet Jackson

2.
Unique ID: JJ234
Song Name: All For You
Artist Name: Janet Jackson
Song Length (in seconds): 391

4.
Unique ID: JJ456
Song Name: Black Eagle
Artist Name: Janet Jackson
Song Length (in seconds): 197
```



# Step 11: Implement the "Output total time of playlist" menu option in ExecuteMenu(). (2 pts)

Output the sum of the time of the playlist's songs (in seconds).

Ex:

```
OUTPUT TOTAL TIME OF PLAYLIST (IN SECONDS)
Total time: 1461 seconds
```



LAB ACTIVITY | 31.15.1: LAB*: Program: Playlist | 0 / 27

Current file: main.cpp | Load default template...

```
1  #include <iostream>
2  #include "PlaylistNode.h"
3
4  using namespace std;
5
6  void PrintMenu(const string playlistTitle) {
7     /* Type your code here */
8
9  }
10
11 PlaylistNode* ExecuteMenu(char option, string playlistTitle, PlaylistNode* h
```

```
12    /* Type your code here */
13
14 }
15
```

Develop mode | Submit mode

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.



Enter program input (optional)

If your code requires input values, provide them here.

Run program

Input (from above) → main.cpp (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

History of your effort will appear here once you begin working on this zyLab.

# 31.16 LAB: Warm up: Parsing strings

This section has been set as optional by your instructor.



(1) Prompt the user for a string that contains two strings separated by a comma. (1 pt)

- Examples of strings that can be accepted:
  - Jill, Allen
  - Jill , Allen
  - Jill,Allen

Ex:

```
Enter input string:
Jill, Allen
```

(2) Print an error message if the input string does not contain a comma. Continue to prompt until a valid string is entered. *Note: If the input contains a comma, then assume that the input also contains two strings.* (2 pts)



Ex:

```
Enter input string:
Jill Allen
Error: No comma in string.

Enter input string:
Jill, Allen
```

(3) Extract the two words from the input string and remove any spaces. Store the strings in two separate variables and output the strings. (2 pts)

Ex:

```
Enter input string:
Jill, Allen
First word: Jill
Second word: Allen
```

(4) Using a loop, extend the program to handle multiple lines of input. Continue until the user enters q to quit. (2 pts)

Ex:

```
Enter input string:
Jill, Allen
First word: Jill
Second word: Allen

Enter input string:
Golden , Monkey
First word: Golden
Second word: Monkey
```

```
Enter input string:
Washington,DC
First word: Washington
Second word: DC

Enter input string:
q
```



539740.3879454.qx3zqy7

**LAB ACTIVITY** 31.16.1: LAB: Warm up: Parsing strings 0 / 7

main.cpp Load default template...

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  int main() {
6
7     /* Type your code here. */
8
9     return 0;
10 }
```

**Develop mode** | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.



**Run program** Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

```
History of your effort will appear here once you begin working
on this zyLab.
```



# 31.17 LAB*: Program: Data visualization

This section has been set as optional by your instructor.

(1) Prompt the user for a title for data. Output the title. (1 pt)

Ex:

```
Enter a title for the data:
Number of Novels Authored
You entered: Number of Novels Authored
```

(2) Prompt the user for the headers of two columns of a table. Output the column headers. (1 pt)

Ex:

```
Enter the column 1 header:
Author name
You entered: Author name

Enter the column 2 header:
Number of novels
You entered: Number of novels
```



(3) Prompt the user for data points. Data points must be in this format: *string, int*. Store the information before the comma into a string variable and the information after the comma into an integer. The user will enter `-1` when they have finished entering data points. Output the data points. Store the string components of the data points in a vector of strings. Store the integer components of the data points in a vector of integers. (4 pts)

Ex:

```
Enter a data point (-1 to stop input):
Jane Austen, 6
Data string: Jane Austen
Data integer: 6
```



(4) Perform error checking for the data point entries. If any of the following errors occurs, output the appropriate error message and prompt again for a valid data point.

- If entry has no comma
  - Output: `Error: No comma in string.` (1 pt)
- If entry has more than one comma
  - Output: `Error: Too many commas in input.` (1 pt)
- If entry after the comma is not an integer
  - Output: `Error: Comma not followed by an integer.` (2 pts)

Ex:

```
Enter a data point (-1 to stop input):
Ernest Hemingway 9
Error: No comma in string.

Enter a data point (-1 to stop input):
Ernest, Hemingway, 9
Error: Too many commas in input.

Enter a data point (-1 to stop input):
Ernest Hemingway, nine
Error: Comma not followed by an integer.

Enter a data point (-1 to stop input):
Ernest Hemingway, 9
Data string: Ernest Hemingway
Data integer: 9
```



(5) Output the information in a formatted table. The title is right justified with a setw() value of 33. Column 1 has a setw() value of 20. Column 2 has a setw() value of 23. (3 pts)

Ex:

```
        Number of Novels Authored
Author name          |       Number of novels
-----------------------------------------
Jane Austen          |                    6
Charles Dickens      |                   20
Ernest Hemingway     |                    9
Jack Kerouac         |                   22
F. Scott Fitzgerald  |                    8
Mary Shelley         |                    7
Charlotte Bronte     |                    5
Mark Twain           |                   11
Agatha Christie      |                   73
Ian Flemming         |                   14
Stephen King         |                   54
Oscar Wilde          |                    1
```



(6) Output the information as a formatted histogram. Each name is right justified with a setw() value of 20. (4 pts)

Ex:

```
         Jane Austen ******
     Charles Dickens ********************
    Ernest Hemingway *********
        Jack Kerouac **********************
 F. Scott Fitzgerald ********
        Mary Shelley *******
    Charlotte Bronte *****
          Mark Twain ***********
     Agatha Christie
**********************************************************************
        Ian Flemming **************
        Stephen King
**************************************************
         Oscar Wilde *
```



539740.3879454.qx3zqy7

**LAB ACTIVITY** | 31.17.1: LAB*: Program: Data visualization | 0 / 17

main.cpp — Load default template...

```
#include <iostream>
#include <string>
#include <vector>
//FIXME: stringstream library
//FIXME: stream manipulation library
using namespace std;

int main() {

   /* Type code here. */

   return 0;
}
```



**Develop mode** | **Submit mode**

Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first box, then click **Run program** and observe the program's output in the second box.

Enter program input (optional)

If your code requires input values, provide them here.

**Run program**

Input (from above) → **main.cpp** (Your program) → Outp

Program output displayed here

Coding trail of your work [What is this?](#)

History of your effort will appear here once you begin working on this zyLab.