27.1 LAB: BankAccount class

This section has been set as optional by your instructor.

Given main(), define the BankAccount class (in files BankAccount.h and BankAccount.cpp) that manages checking and savings accounts. The class has three private data members:

- customer name (string)
- savings account balance (double)
- checking account balance (double)

Implement the following constructor and public member functions as listed below:

- `BankAccount(string newName, double chBalance, double sBalance)` - set the customer name to parameter `newName`, set the checking account balance to parameter `chBalance` and set the savings account balance to parameter `sBalance`.
- `void SetName(string newName)` - set the customer name to parameter `newName`
- `string GetName()` - return the customer name
- `void SetChecking(double balance)` - set the checking account balance to parameter `balance`
- `double GetChecking()` - return the checking account balance
- `void SetSavings(double balance)` - set the savings account balance to parameter `balance`
- `double GetSavings()` - return the savings account balance
- `void DepositChecking(double amt)` - add parameter `amt` to the checking account balance (only if positive)
- `void DepositSavings(double amt)` - add parameter `amt` to the savings account balance (only if positive)
- `void WithdrawChecking(double amt)` - subtract parameter `amt` from the checking account balance (only if positive)
- `void WithdrawSavings(double amt)` - subtract parameter `amt` from the savings account balance (only if positive)
- `void TransferToSavings(double amt)` - subtract parameter `amt` from the checking account balance and add to the savings account balance (only if positive)

LAB ACTIVITY | 27.1.1: LAB: BankAccount class | 0 / 10

Current file: BankAccount.h ▾ Load default template...

```
1  #ifndef BANKACCOUNTH
2  #define BANKACCOUNTH
3
4  #include <string>
5  using namespace std;
6
7  class BankAccount {
8     public:
9        // TODO: Declare public member functions
10
11    private:
12       // TODO: Declare private data members
13 };
14
15 #endif
```


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) → **BankAccount.h** (Your program) → Ou

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.

27.2 LAB: Swap two numbers

This section has been set as optional by your instructor.

Complete the Swap() function in main.cpp to exchange the values of the num field of two Number objects, num1 and num2. Since both num1 and num2 are changed inside Swap(), reference operators (&) are used to pass num1 and num 2 by reference.

Hint: Refer to the given Number class to see the definitions of num field and other functions available.

Ex: If num1 is 19 and num2 is 178, calling Swap(num1, num2) will swap the values so that num1 becomes 178 and num2 becomes 19.

LAB ACTIVITY | 27.2.1: LAB: Swap two numbers | 0 / 10

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

```
1  #include "Number.h"
2  #include <iostream>
3  using namespace std;
4
5  void Swap(Number& num1, Number& num2) {   // num1 and num2 are passed by ref
6     /* Type your code here */
7  }
8
9  int main() {
10    Number num1 = Number(19);
11    Number num2 = Number(178);
12
13    Swap(num1, num2);
14    cout << "num1 = " << num1.GetNum() << ", num2 = " << num2.GetNum() << end
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.
```

27.3 LAB: Rolling for a pair

This section has been set as optional by your instructor.

Given two GVDie objects that represent 2 six-sided dice and an integer that represents a desired value as parameters, complete the function RollingForPair() in the main class. The main class rolls the dice until a pair with the desired value is rolled. The function RollingForPair() then returns the number of rolls thrown to achieve the result. Assume the desired value received from input is within the appropriate range, 1-6.

Note: For testing purposes, the GVDie objects are created in the main() function using a pseudo-random number generator with a fixed seed value. The program uses a seed value of 15 during development, but when submitted, a different seed value will be used for each test case.

Ex: If the GVDie objects are created with a seed value of 15 and the input of the program is:


```
2
```

the output is:

```
It took 82 rolls to get a pair of 2's.
```

LAB ACTIVITY | 27.3.1: LAB: Rolling for a pair | 0 / 10

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

```
#include <iostream>
#include "GVDie.h"
using namespace std;

int RollingForPair(GVDie d1, GVDie d2, int val) {
   /* Type your code here */
}

int main() {
   GVDie d1 = GVDie();
   GVDie d2 = GVDie();
   d1.SetSeed(15); // Create a GVDie object with seed value 15
   d2.SetSeed(15); // Create a GVDie object with seed value 15
   int rolls;
   int val;
```


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.

27.4 LAB: Rolling for X

This section has been set as optional by your instructor.

In main.cpp, complete the function RollSpecificNumber() that takes in three parameters: a GVDie object, an integer representing a desired face number of a die, and an integer representing the goal amount of times to roll the desired face number. The function RollSpecificNumber() then rolls the die until the desired face number is rolled the goal amount of times and returns the number of rolls required.

Note: For testing purposes, the GVDie objects are created in the main() function using a pseudo-random number generator with a fixed seed value. The program used during development uses a seed value of 15, but when submitted, different seed values will be used for each test case.

Ex: If the GVDie objects are created with a seed value of 15 and the input of the program is:

```
3 20
```

the output is:

```
It took 140 rolls to get a "3" 20 times.
```


LAB ACTIVITY | 27.4.1: LAB: Rolling for X | 0 / 10

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

```
1 Loading latest submission...
```

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.
```

27.5 LAB: Calculator class

This section has been set as optional by your instructor.

Given main(), complete the Calculator class (in files Calculator.h and Calculator.cpp) that emulates basic functions of a calculator: add, subtract, multiple, divide, and clear. The class has one private data member called `value` for the calculator's current value. Implement the following constructor and public member functions as listed below:

- Calculator() - default constructor to set the data member to 0.0
- void Add(double val) - add the parameter to the data member
- void Subtract(double val) - subtract the parameter from the data member
- void Multiply(double val) - multiply the data member by the parameter
- void Divide(double val) - divide the data member by the parameter
- void Clear( ) - set the data member to 0.0
- double GetValue( ) - return the data member

Given two double input values num1 and num2, the program outputs the following values:

1. The initial value of the data member, `value`
2. The value after adding num1
3. The value after multiplying by 3
4. The value after subtracting num2
5. The value after dividing by 2
6. The value after calling the clear() method

Ex: If the input is:

```
10.0 5.0
```

the output is:

```
0.0
10.0
30.0
25.0
12.5
0.0
```


LAB ACTIVITY | 27.5.1: LAB: Calculator class | 0 / 10

File is marked as read only Current file: main.cpp

```
1  #include <iostream>
2  #include <iomanip>
3  #include "Calculator.h"
4  using namespace std;
5
6  int main() {
7     Calculator calc;
8     double num1;
9     double num2;
10
11    cin >> num1;
12    cin >> num2;
13
14    cout << fixed << setprecision(1);
15    // 1. The initial value
```


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.
```

27.6 LAB: Count honors

This section has been set as optional by your instructor.

Students can graduate with honors if their GPA is at or above 3.0. Complete the Course class by implementing the CountHonors() member function, which returns the number of students with a GPA at or above 3.0.

Given classes:

- Class Course represents a course, which contains a vector of Student objects as a course roster. (Type your code in here)
- Class Student represents a classroom student, which has three data members: first name, last name, and GPA.

Hint: Refer to the Student class to explore the available functions that can be used for implementing the CountHonors() function.

Note: For testing purposes, different student values will be used.

Ex. For the following students:

```
Henry Cabot 3.2
Brenda Stern 2.9
Lynda Robison 3.6
Jane Flynn 1.8
```

the output is:

```
Honors count: 2
```


539740.3879454.qx3zqy7

LAB ACTIVITY | 27.6.1: LAB: Count honors | 0 / 10

Current file: **Course.cpp**

```
1 Loading latest submission...
```

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) → **Course.cpp** (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

Retrieving signature

27.7 LAB: Flipping for heads

This section has been set as optional by your instructor.

Given main() and GVCoin class, complete function CountHeads() in main.cpp that counts and returns the number of flips taken to achieve a desired number of heads. Function CountHeads() has a GVCoin object and an integer representing the desired number of heads as parameters. Review the definition of "GVCoin.cpp" by clicking on the orange arrow.

Note: For testing purposes, a GVCoin object is created in the main() function using a pseudo-random number generator with a fixed seed value. The program uses a seed value of 15 during development, but when submitted, a different seed value will be used for each test case.

Ex: If the GVCoin object is created with a seed value of 15 and the desired number of heads is 20, then the function CountHeads() returns 40 and the program outputs:

```
Total number of flips for 20 heads: 40
```


LAB ACTIVITY | 27.7.1: LAB: Flipping for heads 0 / 10

Current file: main.cpp

```
1 Loading latest submission...
```

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?

Retrieving signature

27.8 LAB: Simple car

This section has been set as optional by your instructor.

Given two integers that represent the miles to drive forward and the miles to drive in reverse as user inputs, create a SimpleCar object that performs the following operations:

- Drives input number of miles forward
- Drives input number of miles in reverse
- Honks the horn
- Reports car status

Review the definition of "SimpleCar.cpp" by clicking on the orange arrow.

Ex: If the input is:

```
100 4
```

the output is:

```
beep beep
Car has driven: 96 miles
```


LAB ACTIVITY | 27.8.1: LAB: Simple car | 0 / 10

Current file: SimpleCar.cpp

```
1 Loading latest submission...
```

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) → **SimpleCar.cpp** (Your program) → Out

Program output displayed here

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

Retrieving signature

27.9 LAB: Student class

This section has been set as optional by your instructor.

In the Student.cpp file and Student.h file, build the Student class with the following specifications:

- Private data members
  - string name - Initialized in default constructor to "Louie"
  - double gpa - Initialized in default constructor to 1.0
- Default constructor
- Public member functions
  - SetName() - sets the student's name
  - GetName() - returns the student's name
  - SetGPA() - sets the student's GPA
  - GetGPA() - returns the student's GPA

Ex. If a new Student object is created, the default constructor sets name to "Louie" and gpa to 1. The output of GetName() and GetGPA() before and after calling SetName("Felix") and SetGPA(3.7) is:

```
Louie/1
Felix/3.7
```

539740.3879454.qx3zqy7

LAB ACTIVITY 27.9.1: LAB: Student class 0 / 10

Current file: main.cpp

```
1 Loading latest submission...
```

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?

Retrieving signature

27.10 LAB: Product class

This section has been set as optional by your instructor.

Given main() and Product.h, define the Product class (in Product.cpp) that will manage product inventory. Product class has three private data members: a product code (string), the product's price (double), and the number count of product in inventory (int).

Implement the following Constructor and member functions listed in the Product.h file:

- Product(string code, double price, int count) - set the data members using the three parameters
- void SetCode(string code) - set the product code (i.e. SKU234) to parameter code
- string GetCode() - return the product code
- void SetPrice(double p) - set the price to parameter p
- double GetPrice() - return the price
- void SetCount(int num) - set the number of items in inventory to parameter num
- int GetCount() - return the count
- void AddInventory(int amt) - increase inventory by parameter amt
- void SellInventory(int amt) - decrease inventory by parameter amt

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.

Ex. If a new Product object is created with code set to "Apple", price set to 0.40, and the count set to 3, the output is:

```
Name: Apple
Price: 0.40
Count: 3
```

Ex. If 10 apples are added to the Product object's inventory, but then 5 are sold, the output is:

```
Name: Apple
Price: 0.40
Count: 8
```

Ex. If the Product object's code is set to "Golden Delicious", price is set to 0.55, and count is set to 4, the output is:

```
Name: Golden Delicious
Price: 0.55
Count: 4
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 27.10.1: LAB: Product class

0 / 10

File is marked as read only Current file: main.cpp

```
1 Loading latest submission...
```

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?

Retrieving signature

27.11 LAB: Course size

This section has been set as optional by your instructor.

Complete the Course class by implementing the CourseSize() member function, which returns the total number of students in the course.

Given classes:

- Class Course represents a course, which contains a vector of Student objects as a course roster. (Type your code in here.)
- Class Student represents a classroom student, which has three data members: first name, last name, and GPA.

Note: For testing purposes, different student values will be used.

Ex. For the following students:

```
Henry Bendel 3.6
Johnny Min 2.9
```

the output is:

```
Course size: 2
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 27.11.1: LAB: Course size | 0 / 10

Current file: **Course.cpp**

```
1 Loading latest submission...
```


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) → Course.cpp (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

Retrieving signature

27.12 LAB: Dean's list

This section has been set as optional by your instructor.

A student makes the Dean's list if their GPA is 3.5 or higher. Complete the Course class by implementing the GetDeansList() member function, which returns a vector of students with a GPA of 3.5 or higher.

Given classes:

- Class Course represents a course, which contains a vector of Student objects as a course roster. (Type your code in here.)
- Class Student represents a classroom student, which has three data members: first name, last name, and GPA. (Hint: getGPA() returns a student's GPA.)

Note: For testing purposes, different student values will be used.

Ex. For the following students:


```
Henry Nguyen 3.5
Brenda Stern 2.0
Lynda Robison 3.2
Sonya King 3.9
```

the output is:

```
Dean's list:
Henry Nguyen (GPA: 3.5)
Sonya King (GPA: 3.9)
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 27.12.1: LAB: Dean's list | 0 / 10

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

```
1  #include <iostream>
2  #include "Course.h"
3  using namespace std;
4
5  vector<Student> Course::GetDeansList() {
6     /* Type your code here */
7  }
8
9  void Course::AddStudent(Student s) {
10    roster.push_back(s);
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) → Course.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.
```

27.13 LAB: How many dice rolls?

This section has been set as optional by your instructor.

Given a GVDie object and an integer that represents the total sum desired as parameters, complete the function RollTotal() in main.cpp that returns the number of rolls needed to achieve at least the total sum.

Note: For testing purposes, the GVDie object is created in the main() function using a pseudo-random number generator with a fixed seed value. The program uses a seed value of 15 during development, but when submitted, a different seed value will be used for each test case.

Ex: If the GVDie object is created with a seed value of 15 and the input of the program is:

```
20
```

the output is:

```
Number of rolls to reach at least 20: 5
```


LAB ACTIVITY | 27.13.1: LAB: How many dice rolls? 0 / 10

Current file: main.cpp

```
1 Loading latest submission...
```

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?

Retrieving signature

27.14 LAB: Random values

This section has been set as optional by your instructor.

Given main(), complete class RandomNumbers (in files RandomNumbers.h and RandomNumbers.cpp) that has three integer data members: var1, var2, and var3. Write a getter function for each variable: getVar1(), getVar2() and getVar3(). Then write the following 2 member functions:

- SetRandomValues( ) - accepts a low and high integer values as parameters, and sets var1, var2, and var3 to random numbers (generated using the random function) within the range of the low and high input values (inclusive).

- GetRandomValues( ) - prints out the 3 random numbers in the format: "Random values: var1 var2 var3"

Ex: If the input is:

```
15 20
```

the output is:


```
Random values: 17 15 19
```

where 17, 15, 19 can be any random numbers within 15 and 20 (inclusive).

Note: When submitted, your program will be tested against different input range to verify if the three randomly generated values are within range.

LAB ACTIVITY | 27.14.1: LAB: Random values | 0 / 10

Current file: **RandomNumbers.cpp**

```
1 Loading latest submission...
```

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) → RandomNumbers.cpp (Your program)

Program output displayed here

Coding trail of your work What is this?

Retrieving signature

27.15 LAB: Drop student

This section has been set as optional by your instructor.

Complete the Course class by implementing the DropStudent() member function, which removes a student (by last name) from the course roster. If the student is not found in the course roster, no student should be dropped. Assume all students have distinct last names.

Given classes:

- Class Course represents a course, which contains a vector of Student objects as a course roster. (Type your code in here.)
- Class Student represents a classroom student, which has three private data members: first name, last name, and GPA. (Hint: getLast() returns the last name field.)

Note: For testing purposes, different student values will be used.

Ex. For the following students:


```
Henry Nguyen 3.5
Brenda Stern 2.0
Lynda Robison 3.2
Sonya King 3.9
```

the output for dropping Stern is:

```
Course size: 4 students
Course size after drop: 3 students
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 27.15.1: LAB: Drop student | 0 / 10

Current file: **Course.cpp** | **Load default template...**

```
1 Loading latest submission...
```

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) → **Course.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.
```

27.16 LAB: Find student with highest GPA

This section has been set as optional by your instructor.

Complete the Course class by implementing the FindStudentHighestGpa() member function, which returns the Student object with the highest GPA in the course. Assume that no two students have the same highest GPA.

Given classes:

- Class Course represents a course, which contains a vector of Student objects as a course roster. (Type your code in here.)
- Class Student represents a classroom student, which has three private data members: first name, last name, and GPA. (Hint: GetGPA() returns a student's GPA.)

Note: For testing purposes, different student values will be used.

Ex. For the following students:

```
Henry Nguyen 3.5
Brenda Stern 2.0
Lynda Robison 3.2
Sonya King 3.9
```

the output is:

```
Top student: Sonya King (GPA: 3.9)
```

539740.3879454.qx3zqy7

LAB ACTIVITY 27.16.1: LAB: Find student with highest GPA

0 / 10

Current file: Course.cpp

```
1 Loading latest submission...
```

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) → **Course.cpp** (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

Retrieving signature

27.17 LAB: Print student roster

This section has been set as optional by your instructor.

Complete Course.cpp by implementing the PrintRoster() function, which outputs a list of all students enrolled in a course and also the total number of students in the course.

Given files:

- main.cpp - contains the main function for testing the program.
- Course.cpp - represents a course, which contains a vector of Student objects as a course roster. (Type your code in here)
- Course.h - header file for the Course class.
- Student.cpp - represents a classroom student, which has three data members: first name, last name, and GPA.
- Student.h - header file for the Student class.

Ex: If the program has 4 students enrolled in the course, the output looks like:

```
Henry Cabot (GPA: 3.5)
Brenda Stern (GPA: 2.1)
Jane Flynn (GPA: 3.9)
Lynda Robison (GPA: 3.2)
Students: 4
```

539740.3879454.qx3zqy7

LAB ACTIVITY | 27.17.1: LAB: Print student roster | 0 / 10

Current file: **Course.cpp**

```
1 Loading latest submission...
```


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) → Course.cpp (Your program) → Outp

Program output displayed here

Coding trail of your work What is this?

Retrieving signature

27.18 LAB: Flipping for tails

This section has been set as optional by your instructor.

Given main() and GVCoin class, complete function FlipForTails() in main.cpp that counts and returns the number of flips taken to achieve a desired number of tails. Function FlipForTails() has a GVCoin object and an integer representing the desired number of tails as parameters. Review the GVCoin.h header file by clicking on the orange arrow.

Note: For testing purposes, a GVCoin object is created in the main() function using a pseudo-random number generator with a fixed seed value. The program uses a seed value of 15 during development, but when submitted, a different seed value will be used for each test case.

Ex: If the GVCoin object is created with a seed value of 15 and the desired number of tails is 100, then the function FlipForTails() returns 213 and the program outputs:

```
Total number of flips for 100 tails: 213
```


539740.3879454.qx3zqy7

LAB ACTIVITY | 27.18.1: LAB: Flipping for tails 0 / 10

Current file: main.cpp

```
1 Loading latest submission...
```


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?

Retrieving signature

27.19 LAB: Find the maximum in a vector

This section has been set as optional by your instructor.

Complete the FindMax() member function in Numbers.cpp that returns the largest value in vector nums.

Ex: If the vector is:

```
[677, 299, 899, 871, 325, 443, 586, 97, 600, 153]
```

the output is:


```
677 299 899 871 325 443 586 97 600 153
899
```

Note: During development, vector nums is filled with 10 pseudo-random integers in main() using the FillRandomly() member function from Numbers.cpp with a seed value of 7. When submitted, different seed values will be used to generate vectors of different size for the test cases. Refer to the textbook section on Random numbers to learn more about pseudo-random numbers.

539740.3879454.qx3zqy7

LAB ACTIVITY | 27.19.1: LAB: Find the maximum in a vector | 0 / 10

Current file: **Numbers.cpp** Load default template...

```
1  #include <iostream>
2  #include <cstdlib>
3  #include "Numbers.h"
4  using namespace std;
5
6  void Numbers::SetNums(vector<int> nums) {
7     this->nums = nums;
8  }
9
10 vector<int> Numbers::GetNums() {
11    return nums;
12 }
13
14 int Numbers::FindMax() {
15    /* Type your code here. */
```


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) → Numbers.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.
```

27.20 LAB: Consecutive heads

This section has been set as optional by your instructor.

Given main() and GVCoin class, complete method ConsecutiveHeads() in main() that counts and returns the number of flips taken to achieve a desired number of consecutive heads without a tails. Function ConsecutiveHeads() has a GVCoin object and an integer representing the desired number of consecutive heads without a tails as parameters. Review the definition of "GVCoin.cpp" by clicking on the orange arrow.

Note: For testing purposes, a GVCoin object is created in main() using a pseudo-random number generator with a fixed seed value. The program uses a seed value of 15 during development, but when submitted, a different seed value will be used for each test case.

Ex: If the GVCoin object is created with a seed value of 15 and the desired number of consecutive heads is 5, then the function ConsecutiveHeads() returns 53 and the program outputs:


```
Total number of flips for 5 consecutive heads: 53
```


LAB ACTIVITY | 27.20.1: LAB: Consecutive heads 0 / 10

Current file: main.cpp

```
1 Loading latest submission...
```


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?

Retrieving signature

27.21 LAB*: Program: Lucky 7

This section has been set as optional by your instructor.

Program Specifications Write a program to play an automated dice game that uses two dice (GVDie class provided). The player rolls both dice and either wins one credit, loses one credit, or sets a goal for future rolls. The current round ends when player wins or loses a credit. The game ends when credits are zero.

Note: this program is designed for *incremental development*. Complete each step and submit for grading before starting the next step. Only a portion of tests pass after each step but confirm progress.

Step 0. Read starter template and *do not* change the provided code. Two GVDie objects are created. A random seed is read from input and used to initialize a sequence of random numbers. This supports automated testing and creates predictable results that would otherwise be random. Starting credits is read from input.

Step 1 (3 pts). Roll both dice. Player wins one credit by rolling 7 or 11. Player loses one credit by rolling 2, 3, or 12. Otherwise, credits do not change and the player's goal is set to the dice total. The player's goal must be repeated in Step #2 to win a credit. The GVDie class includes Roll() and GetValue() functions. Output dice total and credits. Submit for grading to confirm 3 tests pass.
Ex: If input is:

```
42 10
```

Sample output is:

```
Dice total: 2
Credits: 9
```

Step 2 (4 pts). If player did not win or lose in step 1 (i.e. a goal was set), continue rolling both dice until one of two outcomes: 1) player rolls a 7 and loses one credit or 2) player rolls the goal value and wins one credit. Current round ends. Set goal to -1 and output credits. Submit for grading to confirm 5 tests pass.
Ex: If input is:


```
57 7
```

Sample output is:

```
Dice total: 9
Dice total: 8
Dice total: 7
Credits: 6
```

Step 3 (3 pts). Add a loop to repeat steps 1 and 2 until credits are zero. Count and output the number of rounds when game is over. Submit for grading to confirm all tests pass.
Ex: If input is:

```
73 4
```

Sample output *ends* with:


```
Dice total: 11
Dice total: 7
Credits: 2
Dice total: 5
Dice total: 10
Dice total: 7
Credits: 1
Dice total: 3
Credits: 0
Rounds: 8
```


LAB ACTIVITY | 27.21.1: LAB*: Program: Lucky 7 | 0 / 10

Current file: main.cpp

```
1 Loading latest submission...
```


Develop mode | Submit mode

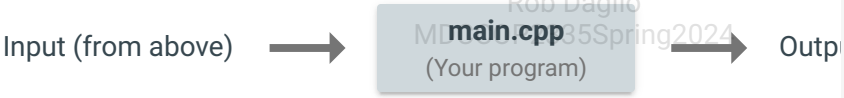
Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first

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?

Retrieving signature

27.22 LAB*: Program: Fancy car

This section has been set as optional by your instructor.

Program Specifications Write a FancyCar class to support basic operations such as drive, add gas, honk horn, and start engine. FancyCar.h declares the functions necessary to complete the exercise. FancyCar.cpp provides the function stubs. Follow each step to gradually complete all functions in FancyCar.cpp.

Note: This program is designed for *incremental development*. Complete each step and submit for grading before starting the next step. Only a portion of tests pass after each step but confirm progress. The main() function includes basic function calls. Add statements in main() as functions are completed to support development mode testing.

Step 0. In FancyCar.h, declare private members for miles driven as shown on the odometer (int), gallons of gas in tank (double), miles per gallon or MPG (double), driving capacity (double), and car model (string). Note the provided constant variable indicates the gas tank capacity of 14.0 gallons.

Step 1 (2 pts). 1) Complete the default constructor by initializing the odometer to five miles, tank is full of gas, miles per gallon is 24.0, and the model is "Old Clunker". 2) Complete the second constructor by passing the model (string) and miles per gallon (double), and initializing all other private members the

same as the default constructor. 3) Complete the accessor functions to check the odometer, check the gas guage, get the model, and get the miles per gallon. Submit for grading to confirm 2 tests pass.

Step 2 (1 pt). Complete the HonkHorn() function to output the car model. If the car model is:

```
Honda Civic
```

the HonkHorn() function outputs:


```
The Honda Civic says beep beep!
```

Submit for grading to confirm 3 tests pass.

Step 3 (1 pt). Complete the Drive() function. Miles driven should increase by parameter distance, and the amount of gas should decrease by distance / MPG. Variables are only updated if parameter distance is positive. Submit for grading to confirm 4 tests pass.

Step 4 (2 pts). Complete the AddGas() function. Increase gas tank by parameter amount only if parameter is positive. Set gas tank to FULL_TANK if too much gas was added. Submit for grading to confirm 5 tests pass.

Step 5 (2 pts). Update the Drive() function to identify if the car runs out of gas. If so, the parameter distance will not be achieved, and the gas tank will have 0.0 gallons. Ex: Drive(100) will not be possible with only three gallons of gas and MPG of 20.0. The maximum driving distance is 60 miles with three gallons of gas and MPG of 20.0. Therefore, the odometer will only increase by 60 instead of the requested 100, and the gas tank will have 0.0 gallons (not a negative amount). Submit for grading to confirm 6 tests pass.

Step 6 (2 pts). 1) Add a boolean private data member in FancyCar.h to indicate if the car engine is on or off. 2) Complete the StartEngine() function to set the boolean variable to true. 3) Complete the StopEngine() function to set the boolean variable to false. 4) Update the constructors to start with the engine off. 5) Update the Drive() function to only update private members if the engine is on, and the engine turns off if the car runs out of gas. 6) Update the AddGas() function to only add gas if the engine is off. Submit for grading to confirm all tests pass.

LAB ACTIVITY | 27.22.1: LAB*: Program: Fancy car | 0 / 10

Current file: main.cpp

```
1 Loading latest submission...
```

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?

Retrieving signature

27.23 LAB*: Program: Self pay kiosk

This section has been set as optional by your instructor.

Program Specifications Write a SelfPayKiosk class to support basic operations such as scan item, cancel transaction, checkout, and make payment. SelfPayKiosk.h declares the functions necessary to complete the exercise. SelfPayKiosk.cpp provides the function stubs. Follow each step to gradually complete all functions in SelfPayKiosk.cpp.

Note: This program is designed for *incremental development*. Complete each step and submit for grading before starting the next step. Only a portion of tests pass after each step but confirm progress. main() in main.cpp includes basic function calls. Add statements as functions are completed to support development mode testing.

Step 0. In SelfPayKiosk.h, declare private data members for number of customers served (int), total sales (double), and current amount due (double). Note the provided constant variable for sales tax of 7%.

Step 1 (1 pt). 1) Complete the constructor to initialize all data members to zero. 2) Complete the accessor functions to return the number of customers served, total sales, and current amount due. Submit for grading to confirm 1 test passes.

Step 2 (2 pts). Complete the ScanItem() function. Increase the amount due by parameter price. Do not update amount due if parameter price is negative. Submit for grading to confirm 3 tests pass.

Step 3 (1 pt). Complete the CheckOut() function. Multiply amount due by SALES_TAX and add to amount due. Submit for grading to confirm 4 tests pass.

Step 4 (2 pts). Complete the MakePayment() function. If parameter payment is enough to pay the amount due, increase total sales by amount due, increment number of customers served, and reset amount due to zero in preparation for the next customer. However, if parameter payment is **not** enough, update total sales by payment and reduce amount due by payment. Do not make any changes if parameter payment is negative. Submit for grading to confirm 6 tests pass.

Step 5 (1 pt). 1) Complete the ResetKiosk() function to reset all data members to zero. 2) Complete the CancelTransaction() function to reset amount due to zero. Submit for grading to confirm 7 tests pass.

Step 6 (2 pts). Complete the SimulateSales() function to perform multiple transactions with increasing prices. Use a loop to simulate parameter numSales transactions. Within the loop, call ScanItem() with parameter initialPrice. Call CheckOut() and MakePayment() to make a payment of $1 more than the amount due. Finally, increase the item price by parameter incrPrice in preparation for the next transaction. Submit for grading to confirm 8 tests pass.

Step 7 (1 pt). Add a boolean data member to indicate if the customer has checked out and is ready to make a payment. Only allow payment after customer has checked out. The CancelTransaction() function should **not** reset amount due if the customer has checked out. Update the following function by inserting assignment statements and if statements related to the new data member: constructor, CheckOut(), MakePayment(), and CancelTransaction(). Ex: Set the boolean data member to false only after full payment has been made. Submit for grading to confirm all tests pass.

LAB ACTIVITY | 27.23.1: LAB*: Program: Self pay kiosk | 0 / 10

Current file: **main.cpp**

```
1 Loading latest submission...
```


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?

Retrieving signature

27.24 LAB: Time difference

The template code defines a Time class. Given a main function that reads two Times and computes the difference between the two, complete the following two functions:

1. `void ReadTime()`

- Reads input for hours, minutes, and seconds, respectively.

2. `void TimeDifference(Time end, Time start)`
   - Compute the amount of time elapsed from start to end. Assume start is always before end.
   - Adjust for negative minutes and seconds.

The main function calls a Time's `ReadTime()` to read input from a user and uses the input to populate the Time's member variables. `PrintTime()` is provided to output Time in 24 hour format (hh:mm:ss).

Ex: If the input is:

```
02 55 10
04 15 50
```

the output is:

```
Start:    02:55:10
End:    04:15:50
Difference: 01:20:40
```


LAB ACTIVITY | 27.24.1: LAB: Time difference | 0 / 10

main.cpp — Load default template...

```
1 Loading latest submission...
```


Develop mode | Submit mode

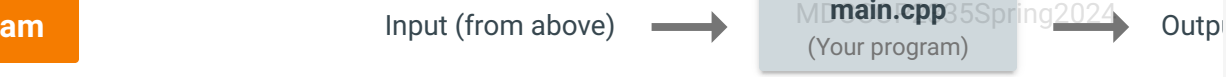
Run your program as often as you'd like, before submitting for grading. Below, type any needed input values in the first

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.
```

27.25 LAB: Resizing shapes (auto)

When assigning a variable's value from the returned value of an overloaded function, the type declaration may be simplified to use the `auto` keyword. However, the `auto` keyword should not be used when the return type of the overloaded function is not already known.

Three shape objects: `Triangle`, `Rectangle`, and `Pentagon`, have private data members for side lengths (integers), the shape's respective constructors, accessors, and print functions. In `main.cpp`, the overloaded `resize()` functions should take one of a Triangle, Rectangle, or Pentagon object, resize the sides to a factor of a non-zero positive integer, and return a new shape object with the resized sides. Ex: A Triangle with side lengths 1, 2, and 3, resized by a factor of 2 should return a new Triangle with sides 2, 4, and 6, respectively.

- Implement the overloaded `resize()` function for each shape.
  - `resize()` takes in a shape object and a positive non-zero integer as the resize factor.
- In each shape's implementation file, implement the shape's respective print function to output the shape's side lengths.

Ex: If a Triangle has side lengths 1, 2, and 3, respectively, the output of `printTri()` is:

```
(1, 2, 3)
```

End each shape's output with a newline.

539740.3879454.qx3zqy7

LAB ACTIVITY | 27.25.1: LAB: Resizing shapes (auto) | 0 / 10

Downloadable files

`main.cpp` , `Triangle.h` , `Triangle.cpp` , `Rectangle.h` , `Rectangle.cpp` , `Pentagon.h` , and `Pentagon.cpp`

Download

Current file: main.cpp

```
1 Loading latest submission...
```

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?

Retrieving signature

©zyBooks 01/31/24 18:15 1939727
Rob Daglio
MDCCOP2335Spring2024