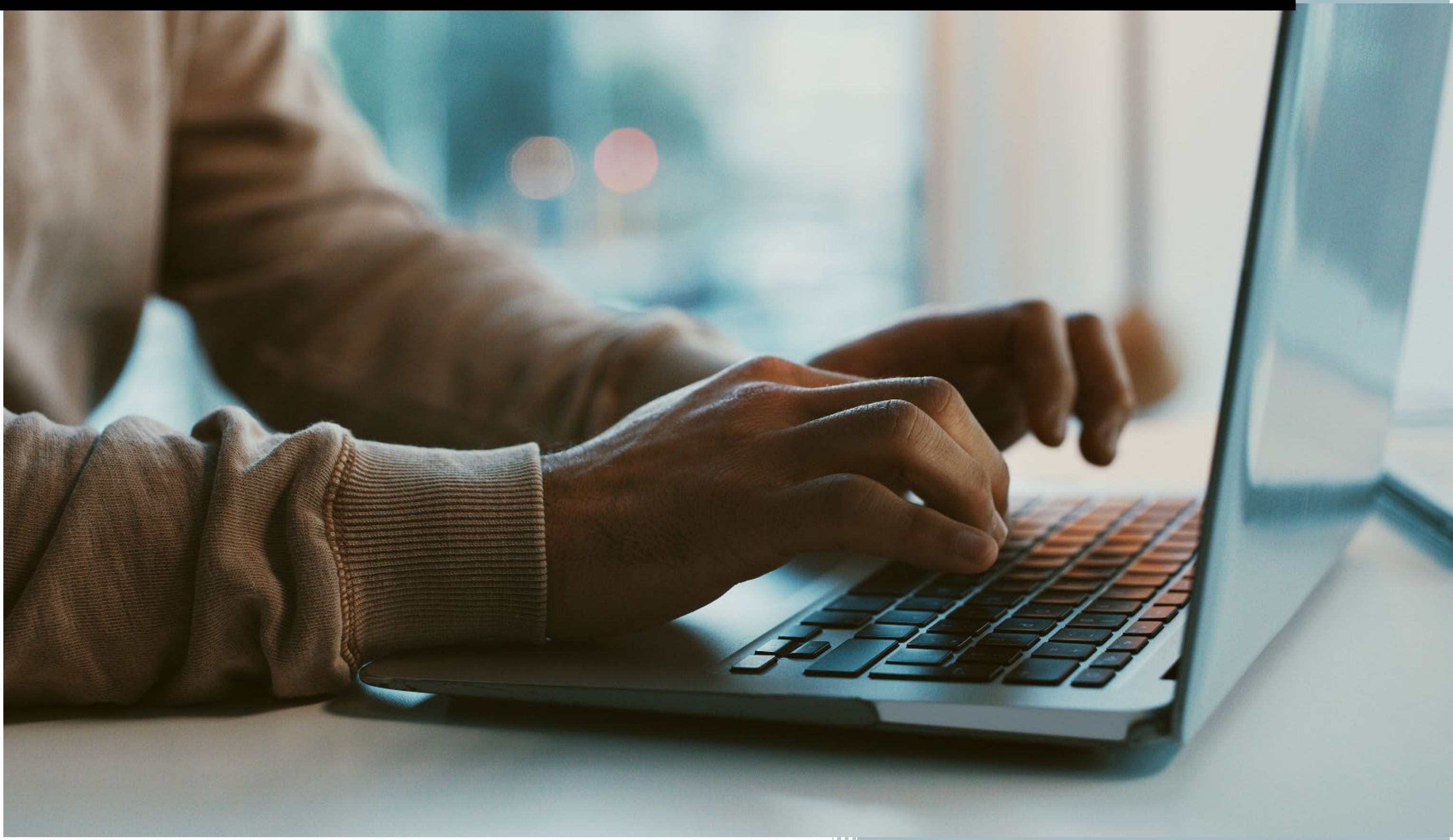
2023

PG2 Topics



Garrett Girod

Full Sail University

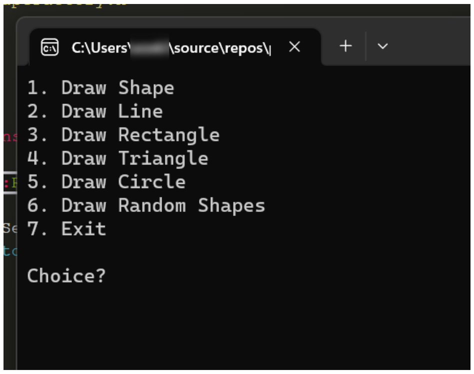
7/1/2023

Contents

Console

The labs will try to enforce a specific window size. However, some systems use the Windows Terminal when debugging which will prevent that code from working correctly. The fix is to **change Windows to use the Console Host** instead of Windows Terminal.

You might experience the problems if your debugger looks like this:

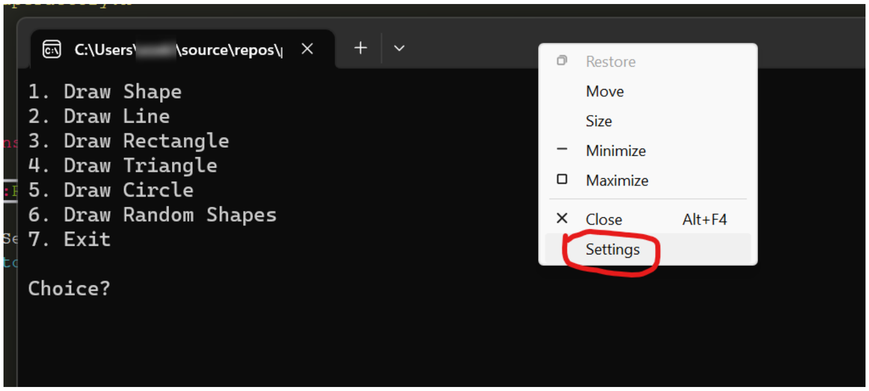


C:\Users\[redacted]\source\repos\

```
1. Draw Shape
2. Draw Line
3. Draw Rectangle
4. Draw Triangle
5. Draw Circle
6. Draw Random Shapes
7. Exit

Choice?
```

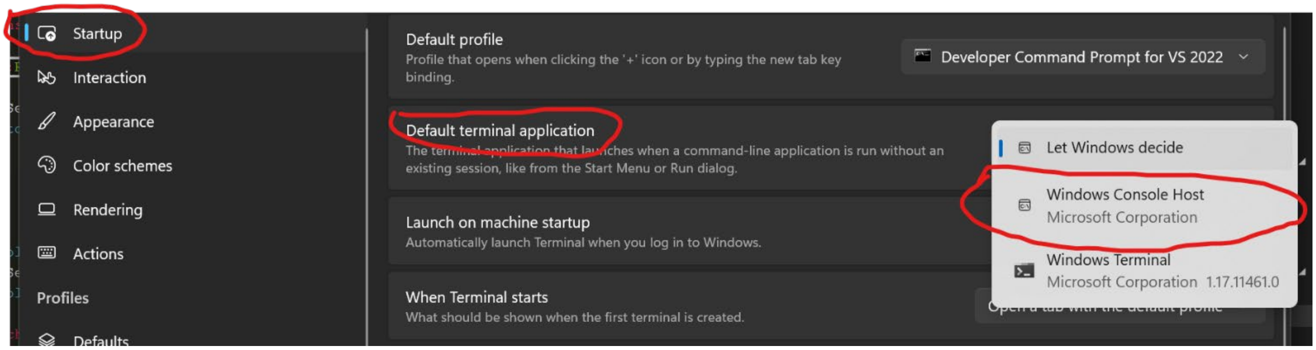
If so, you can right-click and select "Settings":



C:\Users\[redacted]\source\repos\

```
1. Draw Shape
2. Draw Line
3. Draw Rectangle
4. Draw Triangle
5. Draw Circle
6. Draw Random Shapes
7. Exit

Choice?
```



Default profile
Profile that opens when clicking the '+' icon or by typing the new tab key binding.
Developer Command Prompt for VS 2022

Default terminal application
The terminal application that launches when a command-line application is run without an existing session, like from the Start Menu or Run dialog.

Let Windows decide
Windows Console Host
Microsoft Corporation
Windows Terminal
Microsoft Corporation 1.17.11461.0

Launch on machine startup
Automatically launch Terminal when you log in to Windows.

When Terminal starts
What should be shown when the first terminal is created.
Open a tab with the default profile

Also, here's a site describing the different ways: [Windows 11: Change Console to Windows Terminal or Command Prompt (winaero.com)]()

Console Output

std::cout

Links: Basic Input/Output (cplusplus)

std::cout is C++. To use std::cout, you must include the iostream header file.

```
#include <iostream>
```

Use **<<** to output using std::cout.

```
std::cout << "Hello Gotham!\n";
```

You can **output multiple items** on a single line of code.

```
std::cout << "Hello Gotham!\n" << 5 << "\t" << true << "\n";
```

You can modify the output using manipulators like setting the base of the number or forcing uppercase. You can read more about these manipulators here: Manipulators in C++ with Examples - GeeksforGeeks

printf

Links: printf (Programiz)

Links: C++ printf: Printing Formatted Output To Console Or File (marketsplash.com)

printf is C. Include the cstdio header to use it.

```
#include <cstdio>
```

printf uses a format string and an arbitrary length of values.

```
printf("Hello Gotham! Batman's favorite number is %d.\n", 5);
```

You can print multiple variables using the format string.

Example: printing multiple value

```
printf("Hello %s! Batman's favorite number is %d.\n", "Gotham", 5);
```

Input Class

The Input class is provided in the labs to make it easier to get input from the user. You'll find the class in the labs in the **Misc/Input** folders in Solution Explorer. Investigate the **Input.h** file to discover the methods available and how to call them.

There are 4 methods: GetString, GetInteger, GetMenuSelection, and PressEnter. These methods are static therefore to call them you use the Input class name with the :: scope resolution operator.

Examples:

```
std::string myName = Input::GetString("What is your name?");

int age = Input::GetInteger("What is your age?", 0, 120);
```

Notes:

- the first parameter to these methods is the message to show to the user.
- The 2nd and 3rd parameters to the GetInteger method is the min and max range for the integer.

Console Class

The Console class is provided in the labs to make it easier to print things to the Console window. You'll find the class in the labs in the **Misc/Console** folders in Solution Explorer. Investigate the **Console.h** file to discover the methods available and how to call them.

Examples:

```
Console::Write("Hello Gotham.");
Console::Write("Hello Gotham.", ConsoleColor::Yellow, ConsoleColor::Cyan);
Console::WriteLine("Hello Gotham.");
Console::WriteLine("Hello Gotham.", ConsoleColor::Yellow, ConsoleColor::Cyan);
Console::SetCursorLeft(15);
Console::SetCursorPosition(5, 10);
```

Methods

Return types

Lecture Video: Methods

Links: Return Types (W3 Schools)

Specifies the type of the data that is returned from a method or void if the method does not return data.

This method returns an int.

```
int sum(int number1, int number2) { return number1 + number2; }
```

Since it returns int data, assign the returned value to an int variable.

```
int result = sum(5, 13);
```

This method does not return any data.

```
void sayHello(std::string message) { std::cout << "Hello " << message; }
```

Since it does not return data, call the method.

```
sayHello("Gotham");
```

Parameters by value

Lecture Video: Pass by Value

Links: Parameters / Arguments (W3 Schools)

Creates a new variable in the method and copies the value into the new variable from the calling code.

This method has 2 parameters that are passed by value.

```
int sum(int number1, int number2) { return number1 + number2; }
```

The values for number1 and number2 are **copied** into them from the code that calls the method.

```
int sum1 = sum(5, 2);
int sum2 = sum(13, 5);
```

The first line will copy 5 into the number1 variable and copy 2 into the number2 variable. The second line will copy 13 into the number1 variable and copy 5 into the number2 variable.

Parameters by reference

Lecture Video: Pass by Reference

Links: Pass by Reference (W3 Schools)

Creates a reference to the variable used when calling the method instead of making a copy.

```
void makeEven(int& number) { if (number % 2 != 0) ++number; }
```

When calling the method, the variable used when calling the method will be used inside the method.

```
int num = 5;
makeEven(num);
std::cout << num << "\n";
```

The method will modify the num variable and will print 6 instead of 5.

Calling non-static methods

If a method is non-static, then you need to call it on a variable of that class.

Example: to call a non-static method called sum on the Calculator class

```
Calculator t1000; //get a variable of the class
int sum3 = t1000.sum(5, 10); //call the method on the variable
```

Calling static methods

If a method is static, then you use the name of the class to call it.

Example: the reset method in the Console class is static. Therefore, you use the Console name to call the method.

```
Console::reset();
```

Const Methods

Lecture Video: Const Modifiers

Links: Const Methods (GeeksForGeeks)

When adding the const modifier to a method, it means the method cannot modify class member variables.

```
class Sample
{
private:
    int mSomeData = 5;
    int WhatIsTheData() const
    {
        mSomeData += 10;//not allowed to change fields!
        return mSomeData;
    }
};
```

Const Parameters

Lecture Video: Const Modifiers

Links: Const Parameters (DelftStack)

Adding the const modifier to a parameter means that the method is not allowed to modify the value of the parameter.

```
float average(const std::vector<int>& scores)
{
     scores.push_back(5); //not allowed because scores is marked as const
```

Optional (default) Parameters

Lecture Video: Optional Parameters

Links: Optional/Default Parameters (W3Schools)

A default parameter is when a parameter declares what the value of a parameter should be IF the calling code does not provide the value.

```
void spiderVerse(int verse = 616)
{
    std::cout << "Welcome to Earth-" << verse << "\n";
}

spiderVerse(); //the method will use the default value of 616
spiderVerse(67);//the method will use 67 instead of the default value
```

Arrays

Lecture Video: Arrays

Links: C++ Arrays (tutorialspoint.com)

Declaring

When declaring an array on the stack, you must declare how big the array is either with a constant number or with an initialization list.

```
//an array that holds 10 ints.
int highScores[10];

//a string array that holds 3 strings. The compiler can figure out that the size is 3.
std::string JLA[]{ "Batman", "Superman", "Wonder Woman" };
```

Initializing

You can initialize the values in the array using the initializer when declaring the array.

```
std::string JLA[]{ "Batman", "Superman", "Wonder Woman" };
```

Or you can assign values to the spots in the array using the indexer on the left-hand side of the assignment operator.

```
JLA[1] = "Aquaman"; //stores "Aquaman" in the 2nd spot in the array
```

Accessing

To retrieve a value from an array, use the zero-based index with the [ ] indexer.

```
//retrieve the 1st item in the array and assign it to a string variable
std::string theBest = JLA[0]; //assigns "Batman" to theBest
```

To update a value in the array, use the array on the left-hand side of an assignment operator. Use the index in the [ ] to indicate which spot in the array to update.

```
JLA[1] = "Flash"; //updates the 2nd spot in the array
```

Looping

For loops

You must know the size of the array when using a for loop.

```
for (size_t i = 0; i < 3; i++)
   std::cout << JLA[i] << "\n";
```

If it is a stack array, you can calculate the size: sizeof(arrayVariable) / sizeof(type_of_array). EX: if the array is an int array called numbers, sizeof(numbers) / sizeof(int).

```
int nums[]{ 5,4,3,2,1 };
size_t count = sizeof(nums) / sizeof(int);
for (size_t i = 0; i < count; i++)
    std::cout << nums[i] << "\n";
```

Range-based

You can use a range-based for loop to loop over an array:

```
for(std::string name : JLA)
   std::cout << name << "\n";
```

Vectors

To use vectors in your code, you must include the vector header.

```
#include <vector>
```

Lecture Video: Vectors

Links: Vector in C++ STL (GeeksforGeeks)

Declaring

Vector<T> is a template class. To declare a vector variable, you must replace the T with the type of the data you want to store in the vector.

```
        std::vector<int> highScores; //stores ints
        std::vector<std::string> names; //stores strings
```

Presizing

You can presize the vector when creating it so that its internal array has specific size.

```
        scores.reserve(10); //makes the internal array to hold 10 items.
```

Adding Items

You can add items in the **initializer** when you create a vector or you can use the **push_back** method.

```
        std::vector<int> scores { 1, 2, 3, 4 }; //add items on the initializer
        scores.push_back(rand()); //adds a random int to the end of the vector
```

Accessing

Use the indexer to access a specific item in the vector.

```
        int score = scores[0]; //indexes are zero-based like arrays
```

To update a value in the vector, use the [index] on the left-hand side of an assignment operator.

```
        scores[0] = 10; //stores 10 at the first index
```

Looping

Lecture Video: Looping Vectors

For loops

Use the **size()** method to control the for loop.

Use the for loop variable to access the items in the vector.

```
for (int i = 0; i < scores.size(); ++i)
    std::cout << scores[i] << " ";
```

Iterators

begin() - returns an iterator pointing to the first element

end() - returns an iterator pointing to the element AFTER the last element

```
for (auto i = scores.begin(); i != scores.end(); ++i)
    std::cout << *i << " ";
// *i dereferences the iterator to give you the value at that location
```

Range-based

The first part of the range-based for loop is the variable. Each time through the loop, the next item will be assigned to the variable.

auto will let the compiler determine what the type of the variable is based on the collection. You can also put the type explicitly. In the example below, you can use int instead.

The second part of the range-based for loop is the vector (or collection) you are looping through.

```
for (auto highScore : scores)
    std::cout << highScore << " ";
```

Removing Items

Lecture Video: Removing Items from Vectors

Clearing

The **clear()** method removes all the items in a vector.

```
std::vector<int> scores { 1, 2, 3, 4 };
scores.clear();
```

Removing Item

The **erase(position)** method removes the item at the position.

- Position is an iterator.
- Use **begin()** to start from the beginning of the vector.
    - `scores.erase(scores.begin());`will remove the first item in the vector.
- Add a number to **begin()** to remove an item at a different index.
    - `scores.erase(scores.begin() + 2);`will remove the 3rd item in the vector.

Removing Range

The **erase(start, end)** method removes the items in the iterator range.

- **start** and **end** are iterators.
- The item at the end position is **not** removed.
- `scores.erase(scores.begin() + 2, scores.begin() + 4);` will remove the 3rd and 4th items.

Removing in a loop

Lecture Video: Removing in a Loop

When removing an item, every item to the right of the item being removed is shifted down 1 spot in the vector. If you remove items in a loop, then you need to make sure that items are not going to be skipped.

Removing in a for loop.

```
for (size_t i = 0; i < highScores.size();)   {
    if (highScores[i] < 2500)
       highScores.erase(highScores.begin() + i);
    else
       ++i; //only increments the for loop variable if an item is NOT moved
}
```

Removing using std::remove_if

```
highScores.erase(
     std::remove_if(highScores.begin(),
                    highScores.end(),
                    [](int score) { return score < 2500; }),
     highScores.end());
```

Copying Vectors

Lecture Video: Copying Vectors

Copy Manually

```
std::vector<int> scores = { 1,2,3,4,5 };
std::vector<int> scores2;
for (size_t i = 0; i < scores.size(); i++)
    scores2.push_back(scores[i]);
```

Assignment operator

```
std::vector<int> scores = { 1,2,3,4,5 };
std::vector<int> scores3 = scores;
```

Passing a vector to the constructor

```
std::vector<int> scores = { 1,2,3,4,5 };
std::vector<int> scores4(scores);
```

Recursion

Lecture Video: Recursion

Links: Recursion (W3Schools)

A recursive method is a method that calls itself.

```
unsigned long factorial(unsigned int N)
{
    if (N <= 1)  //here's the exit condition!
      return 1;
    return N * factorial(N - 1); //here's the recursive call
}
```

Swap Logic

Lecture Video: Swap Logic

Links: std::swap (cplusplus.com)

Swap with temp variable

```
        void swapper(std::vector<int> vec, int index)
        {
            int temp = vec[index - 1];
            vec[index - 1] = vec[index];
            vec[index] = temp;
        }
```

Swap with std::swap

Include the algorithm header.

```
#include <algorithm>
```

```
        void swapper(std::vector<int> vec, int index)
        {
            std::swap(vec[index - 1], vec[index]);
        }
```

Comparing Strings

Lecture Video: Comparing Strings

Links: Ways to Compare Strings (DigitalOcean.com)

When comparing strings using the methods below, you will get an int value returned.

When comparing string s1 to string s2…

- < 0 means s1 is **LESS THAN** (comes before) s2
- 0 means s1 is **EQUAL TO** s2
- > 0 means s1 is **GREATER THAN** (comes after) s2

_stricmp (MSDN link)

The C way to compare strings…

```
        std::string s1 = "Batman", s2 = "Aquaman";
        int compResult = _stricmp(s1.c_str(), s2.c_str());
```

Performs a case **insensitive** comparison. The strcmp group of functions require a pointer to a char array so you must call c_str on the std::string.

_stricmp returns an int:

- < 0 if s1 is LESS THAN s2
- 0 if s1 is EQUAL TO s2
- > 0 if s1 is GREATER THAN s2

std::string::compare

The C++ way to compare strings…

std::string::compare performs a **case-sensitive** comparison. To do a case-insensitive comparison, make the string uppercase or lowercase before comparing. The method returns the same values as the _stricmp method described above.

```
        std::string s1 = "Batman", s2 = "Aquaman";
        int compareResult = s1.compare(s2);
```

Making Uppercase

```
        std::string best = "Batman";
        for (auto& c : best) c = toupper(c);
```

This will modify the string. If you want to keep the string unmodified, make a copy of the string before making it uppercase.

Maps

To use the std::map data type, you must include the map header.

```
#include <map>
```

Lecture Video: Intro to Maps

Links: stl::map (cppreference)

Declaring

Lecture Video: Declaring and Adding

Links: Maps (GeekForGeeks)

std::map<Tkey,Tvalue>

To declare a map variable, you must replace TKey and TValue with types.

```
        std::map<std::string, double> menuPrices;
```

The keys will be strings and the values will be doubles.

Adding Key-Value Pairs

Lecture Video: Declaring and Adding

Links: Map Insert Method (GeekForGeeks)

Using the Insert method

The Insert method needs a std::pair argument. Use the std::make_pair method to create a std::pair object.

Will return a std::pair where the first part is an iterator pointing to the inserted item and the second part is a bool indicating if the item was inserted (pair<iterator, bool>).

```
        std::map<std::string, double> grades;
        auto gradesInserted = grades.insert(std::make_pair("Bruce", rand() % 101));
        if (gradesInserted.second == false)
           std::cout << "The student was already in the course. \n";
        else
           std::cout << "The student was added.\n";
```

Using [key] = value

Use the assignment operator to assign a value to the key. EX: variable[key] = value;

```
        std::map<std::string, double> menuPrices;
        menuPrices["Curly Fries"] = 3.99;
```

Looping

Lecture Video: Looping

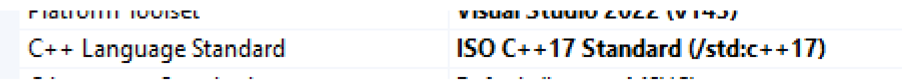
Links: Map Looping (DelftStack)

Using Iterators

```
        std::map<std::string, double>::iterator iter = menuPrices.begin();
        while (iter != menuPrices.end())
        {
              std::string key = iter->first;
              double value = iter->second;
              ++iter;//move the iterator to the next key-value pair in the map
        }
```

Using range-based for loop

This loop requires C++ v17 or higher.



Platform Toolset	Visual Studio 2022 (v143)
C++ Language Standard	ISO C++17 Standard (/std:c++17)

```
        for (const auto& [key, value] : menuPrices)
        {   }
```

Finding Keys

Lecture Video: Finding Keys

Links: Map Finding Keys (DelftStack)

Use the find(key) method to search the map for a key and its value. It will return an iterator. If the iterator is != the end, then the key was found.

```
        std::map<std::string, double>::iterator isFound = menuPrices.find("Nuggets");
        if (isFound != menuPrices.end()) //if true, we found the key
           double value = isFound->second;
        else //the key was not found
           std::cout << "Nuggets was not found.";
```

Updating Values

Lecture Video: Updating Values

Use the assignment operator to update a value for the key. EX: variable[key] = value; If the key is found, the value is overwritten. If the key is not found, the key and value will be added to the map.

```
        menuPrices["Curly Fries"] = 5.99;
```

Removing Items

Lecture Video: Removing Keys and Values

Links: Map Removing Items (GeeksForGeeks)

erase(iterator)

Removes an element at a given position.

```
auto foundKey = menuPrices.find("Mac-n-cheese");
if(foundKey != menuPrices.end())
   menuPrices.erase(foundKey);
```

erase(key)

Returns the number of items that have been removed from the map.

```
int count = menuPrices.erase("Chicken Nuggets");
```

Classes

Classes are usually split into 2 files: a header file (.h) and a source file (.cpp). The header file contains the "declarations" – what describes the class. The source file contains the code for the class.

Lecture Video: Classes

Links: C++ Classes (W3Schools)

Links: Classes and Objects (W3Schools)

Access Modifiers

Lecture Video: Access Modifiers

Links: Access Modifiers (specifiers) (W3Schools)

- Public – members are accessible from outside the class.
- Private – members cannot be accessed outside the class. This is the default if not specified.
- Protected – members cannot be accessed outside the class however they can be accessed in inherited classes.

In C++ classes, the access for members are defined in groups. The access modifier is specified then the items declared after the modifier have that modifier applied to them.

```
class Sample
{
public:
    int x; //public to everyone
private:
    int y; //private to everyone
protected:
    int z; //private to everyone except inherited classes
};
```

Fields

Lecture Video: Fields

Fields are the data of your class. These are just variables defined at the class level, not the method level. Fields are usually private to protect them from modification outside of the class.

In this example, the instance variables x,y,z are defined at the class level and are visible to all instance methods inside the class.

```
class Sample
{
private:
    //these instance variables can be seen by all instance methods in the class
    int x, y, z;
};
```

Methods

Lecture Video: Methods

Links: Class Methods (W3Schools)

Methods are the behaviors of your class (what your class can do). These are sometimes referred to as member functions.

Methods of a class are usually split into the "declaration" in the header file (.h) and the "definition" in the source file(.cpp).

```
class Calculator
{
public:
    //the declaration
    int sum(int number1, int number2); //no code here
};

//the definition
int Calculator::sum(int number1, int number2)
{
    return number1 + number2;
}
```

Getters/Setter methods

Lecture Video: Getters/Setters

Links: Getters/Setters (W3Schools)

The getters/setters are special methods that provide access to the fields of a class. They are the gatekeepers!

The get method returns the value. The set method assigns a value to the field.

```
class Sample
{
private:
    int r_, g_, b_;

public:
    int getRed() { return r_; }
    void setRed(int r) { r_ = r; }
};
```

Constructors

Lecture Video: Constructors

Links: Constructors (W3Schools)

Constructors are special methods that are used to initialize the fields.

- Must have the same name as the class.
- Cannot have a return type.

```
class Color
{
public:
    Color(int r, int g, int b)
    {
        r_ = r; //assign the parameters to the fields
        g_ = g;
        b_ = b;
    }
private:
    int r_, g_, b_;
};
```

Member Initialization Lists

It is preferrable to use member initialization lists to initialize the fields in the constructor.

```
class Color
{
public:
    Color(int r, int g, int b)
        : r_(r), g_(g), b_(b)  //member initialization list
    {  }
private:
    int r_, g_, b_;
};
```

Structs

Lecture Video: Structs

Links: Structs (W3Schools)

Structs are the same as classes except that members are **public** by default.

```
struct COLOR
{
    //public by default
    unsigned short red;
    unsigned short green;
    unsigned short blue;
    unsigned short alpha;
};
```

Inheritance

Inheritance allows you to use another class (the base class) to define a new class.

class <derived_class_name> : <access-specifier> <base_class_name>

```
class FlyingCar : public Car
```

Lecture Video: Inheritance

Links: Inheritance (W3Schools)

Modes of inheritance

- **Public inheritance**: the public members of the base will be public in the derived and protected members of the base will be protected in the derived.
- **Protected inheritance**: both public and protected members of the base will become protected in the derived.
- **Private inheritance**: both public and protected members of the base will become private in the derived.

```
class A
{
public:
    int x;

protected:
    int y;

private:
    int z;
};

class B : public A
{
    //x is public
    //y is protected
    //z is not accessible from B
};
class C : protected A
{
    //x is protected
    //y is protected
    //z is not accessible from C
};
class D : private A //private is the default
{
    //x is private
    //y is private
    //z is not accessible from D
};
```

Constructors

Lecture Video: Inheritance and Constructors

The constructors in the inherited class must call a base class constructor.

```
class Car
{
public:
    Car() :
        mMake("Ford"),
        mModel("A"),
        mModelYear(1908)
    {   }
    Car(int year, std::string make, std::string model) :
        mModelYear(year),
        mMake(make),
        mModel(model)
    {}
private:
    int mModelYear;
    std::string mMake, mModel;
}

class FlyingCar : public Car
{
public:
    FlyingCar()//calls the default constructor of the base
    {}

    FlyingCar(int maxAlt, int yr, std::string make, std::string model)
        : Car(yr, make, model) //calls the base constructor
    {}
};
```

Polymorphism

Overloading

Compile-Time

This kind of polymorphism happens at compile time when the application is built by the compiler.

Lecture Video: Compile-Time Polymorphism

Links: Method Overloading (GeeksForGeeks)

Links: Operator Overloading (GeeksForGeeks)

Method overloading

Multiple methods are defined with the same name but different parameters.

- Different on the number of parameters
  ```
  int add(int n1, int n2);
  int add(int n1, int n2, int n3);
  ```

- Different on the types of the parameters
  ```
  float add(float f1, float f2);
  double add(double d1, double d2);
  ```

Operator overloading

Operator overloading allows you to define how an operator like the plus operator works for your class.

```
class Account
{
private:
    double mBalance;
public:
    Account operator+(Account const& obj)
    {
        Account result;
        result.mBalance = mBalance + obj.mBalance;
        return result;
    }
};
```

Overriding

Runtime

This kind of polymorphism happens at runtime as the application is running. The correct method to call is resolved at runtime.

Lecture Video: Runtime Polymorphism

Links: Virtual Method Overriding (PencilProgrammer)

Overriding

Overriding allows you to redefine how a base class method works for the derived class. This is done in 2 steps:

1. Mark the base class method as virtual.

   ```
   class base
   {
   public:
         virtual void print()
         {
          printf("Hello base.");
      }
   };
   ```

2. Define a new method in the derived class with the same signature as the base class method.
   a. Optionally, you can add the override keyword to the derived class method to provide more compile time checking to enforce that signatures match.

   ```
   class derived : public base
   {
   public:
         //overrides the base version
         void print() override
         {
               printf("Hello derived.");
         }
   };
   ```

Casting

Unique Pointers

Lecture Video: Unique Pointers

Links: std::unique_ptr (Learn C++)

To access std::unique_ptr, you must include the memory header.

```
#include <memory>
```

Use the make_unique method to generate a unique_ptr. You pass arguments to the method that match the arguments needed to call the constructor of the class.

Given these classes…

```
class base
{
public:
    base(int num) : mNumber(num)
    { }
private:
    int mNumber;
};

class derived : public base
{
public:
    derived(std::string str, int num) : base(num), mStr(str)
    { }
private:
    std::string mStr;
};
```

Generate a unique pointer to a derived class instance…

```
std::unique_ptr<derived> pDerived = std::make_unique<derived>("Gotham", 5);
```

Upcasting

Lecture Video: Upcasting

Upcasting is when you point to a derived object through a base type variable. This is always safe because the compiler knows the relationship between these classes.

There are a couple of ways to upcast…

1. Make a unique pointer for the derived class but assign it to a unique_ptr variable of the base type:
   ```
   std::unique_ptr<base> pBase = std::make_unique<derived>("Gotham", 5);
   ```

2. If you already have a unique_ptr to a derived type, then you can use the **std::move** method to upcast.
   ```
   std::unique_ptr<base> pBase = std::move(pDerived);
   ```

Misc. OOP Concepts

Nested Class

Lecture Video: Nested Class

A class defined inside another class.

The nested class can access the private and protected members of the container class.

```
class Enclosing {
private:
    int x;
    /* start of Nested class declaration */
    class Nested {
        int y;
        void NestedFun(Enclosing* e) {
            std::cout << e->x;// works fine: nested class can access
            // private members of Enclosing class
        }
    }; // declaration Nested class ends here
}; // declaration Enclosing class ends here
```

Abstract Class

Lecture Video: Abstract Class

Abstract classes are used ***exclusively as base classes*** for other classes. You cannot create an instance of an abstract class.

You make a class abstract by making at least one method a pure virtual method.

```
class Weapon  //an abstract class
{
public:
    virtual int calcDamage() = 0;//pure virtual
};
```

Static Members

Lecture Video: Static Members

Static Variables

Static variables declared inside a method is allocated once (even if the method is called multiple times) and remains in memory for the lifetime of the application.

```
void demo()
{
    static int count = 0; //created once
    std::cout << count << " ";
    count++;
}
int main()
{
    for (size_t i = 0; i < 10; i++)
        demo();
    //prints 0 1 2 3 4 5 6 7 8 9
}
```

Static Fields

If a **data member (field)** is marked as static, then the variable is **created once** and **remains in memory** for the lifetime of the application. The static data members are **shared** by all instances of the class.

```
class Car
{
public:
    Car(int year) : mModelYear(year)
    {
        mNumberOfCarsMade++;
    }

    //each car has its own model year variable
    int mModelYear;

    //shared by ALL cars
    static int mNumberOfCarsMade;
};
```

You need to initialize the static field **outside of the class methods**.

```
//initialize with class name scoping
int Car::mNumberOfCarsMade = 0;
```

Static Methods

Class methods that are marked as static can ***only access static members*** of the class.

```
static void reporting()
{
    std::cout << "Model year: " << mModelYear << "\n"; //ERROR! cannot access non-static members
    std::cout << "Number of cars made: " << mNumberOfCarsMade << "\n";
}
```

Non-static Methods

Non-Static methods can access static and non-static members of the class. Non-static methods have a **hidden pointer parameter** called **‘this’**. It points to the object that the method was called on. *Only use ‘this->’ in your method to eliminate ambiguity.*

```
    void vehicleInfo() //there's a hidden parameter called 'this’
    {
        std::cout << "Model Year: " << this->mModelYear << "\n";
    }
```

Final Specifier

Lecture Video: Final Specifier

Final specifies that…

- a virtual function cannot be overridden in a derived class or
- that a class cannot be derived from.

Final Methods

```
class rocketEngine : public engine
{
public:
    virtual void ApplyThrust() final
    { }
};
```

Final Classes

A final class prevents other classes from inheriting from it.

```
class rocketEngine final
{
};

class carEngine : public rocketEngine  //BUILD ERROR!
{
};
```

Friends

Lecture Video: Friends

A friend method can access private and protected members of a class. A friend method can be a global method or a class method of another class.

A Global Method

```
class Box
{
private:
    int width, height;
public:
    //grant render function access
    //to my private members
    friend void render(Box& box);
};

//a global method
void render(Box& box)
{
    //accessing the private fields of box
    if (box.width > 0 && box.height > 0)
    {    }
}
```

A Class Method

```
class Box
{
private:
    int width, height;
public:
    //grant GraphicsEngine::Draw function
    //access to my private members
    friend void GraphicsEngine::Draw(Box& box);
};

class GraphicsEngine
{
public:
    void Draw(Box& box);
};

void GraphicsEngine::Draw(Box& box)
{
    //accessing the private fields of box
    if (box.width > 0 && box.height > 0)
    {    }
}
```

File I/O

To access the io stream classes, you must include the fstream header.

```
#include <fstream>
```

Lecture Video: File I/O

Links: Files (W3Schools)

Writing CSV Data

Lecture Video: Writing CSV Data

Use the **ofstream** class to write a CSV file.

Remember to separate the data in the file with a delimiter.

```
        std::ofstream file("myData.csv");
        char delimiter = '|';
        file << "Batman!" << delimiter << 5 << delimiter << 13.7 << "\n";
        file.close();
```

Reading CSV Data

Lecture Video: Reading CSV Data

Use the **ifstream** class to write a CSV file.

You can read a line from the file using the **std::getline** method.

```
        std::ifstream inFile("myData.csv");
        std::string line;
        std::getline(inFile, line); //reads 1 line from the file
        inFile.close();
```

Parsing CSV Data

Lecture Video: Parsing CSV Data

Links: StringStream (Marketsplash)

Even though we wrote out data of different types (string, int, double), when we read it in again, we only get strings. Therefore, we need to parse the string data to get the individual pieces of data.

We can also use **std::getline** to get each piece of data from the line itself. We'll use the stringstream class with the std::getline method to read each piece of data from the string.

To access the stringstream class, you must include the sstream header.

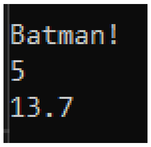
```
#include <sstream>
```

Store the string in a stringstream instance. Then loop over the stringstream instance using getline to read the data.

NOTE: this example assumes the delimiter is '|'.

```
std::stringstream strStream(line);
std::string data;
while (std::getline(strStream, data, '|'))
{
    std::cout << data << "\n";
}
```

The output using the file from the "Writing CSV Data" section:



```
Batman!
5
13.7
```

Serializing

Lecture Video: [Serializing Data]

Serializing is the process of storing object instances to a file or stream. It stores the state of the object (the data of the objects are saved).

Given an object, we take the values of the data members and write them to a file.

```
class Car
{
private:
      int mModelYear;
      std::string mModel;
      std::string mMake;
public:
      void serialize(std::ofstream& outFile, char delimiter)
      {
            outFile << mModelYear << delimiter << mMake << delimiter << mModel;
      }
```

Deserializing

Lecture Video: Deserializing Data

Deserializing loads the state of objects to set the data on new instances in the application.

If we add a deserialize method to the Car class:

```
void deserialize(std::string csvData, char delimiter)
{
    //data format: year make model
    std::string outStr;
    std::stringstream sPart(csvData);
    std::getline(sPart, outStr, delimiter);
    mModelYear = std::stoi(outStr);//converts string to int
    std::getline(sPart, mMake, delimiter);
    std::getline(sPart, mModel, delimiter);
}
```