Project 1: Bag ADT Homework projects should be worked on by each individual student. Brainstorming and sketching out problems on paper or on a whiteboard together are permitted, but do not copy code from someone else or allow your code to be copied. Students who commit or aid in plagiarism will receive a 0% on the assignment and be reported.

Information

Topics: Data structures, array implementations

Turn in: Turn in all source files - .cpp, .hpp, and/or .h files. **Do not turn in Visual Studio files**.

Starter files: Download from GitHub.

Building and running: If you are using Visual Studio, make sure to run with debugging. (Don't run without debugging!) Using the debugger will help you find errors.
To prevent a program exit, use this before `return 0;`

```
cin.ignore();
cin.get();
```

Tips:

- Always make sure it builds. Only add a few lines of code at a time and build after each small change to ensure your program still builds.
- One feature at a time. Only implement one feature (or one function) at a time. Make sure it builds, runs, and works as intended before moving on to the next feature.
- Search for build errors. Chances are someone else has had the same build error before. Copy the message into a search engine and look for information on *why* it occurs, and *how* to resolve it.
- Use debug tools, such as breakpoints, stack trace, and variable watch.
- Don't implement everything in one go. Don't just try typing out all the code in one go without building, running, and testing. It will be much harder to debug if you've tried to program everything all at once.

Contents

PART 1

About

For this first project, we will step through implementing a bag object with a static array. This project will be a bit hand-holdey, so you can get used to the tools and basics of coding in C++.

You can also use Chapter 3 of the textbook for additional information on how to implement these functions.

PART 2

Project setup

Setting up an IDE

IDE stands for "Integrated Development Environment". An IDE is a program that contains the text editor for code, as well as some project management tools and debugging tools.

We will step through how to create a project in both Visual Studio and in Code::Blocks. If you're working on your own computer, you can get a free version of Visual Studio Community Edition. If VS is too bulky or slow, or if you're not running Windows, then Code::Blocks is a good option as well.

Visual Studio: `https://www.visualstudio.com/vs/community/`

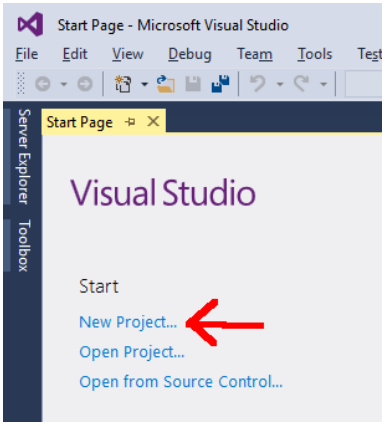
Code::Blocks: `http://codeblocks.org/downloads/26`

Note: If installing for Windows, download the version that says "mingw-setup.exe".

Creating a project

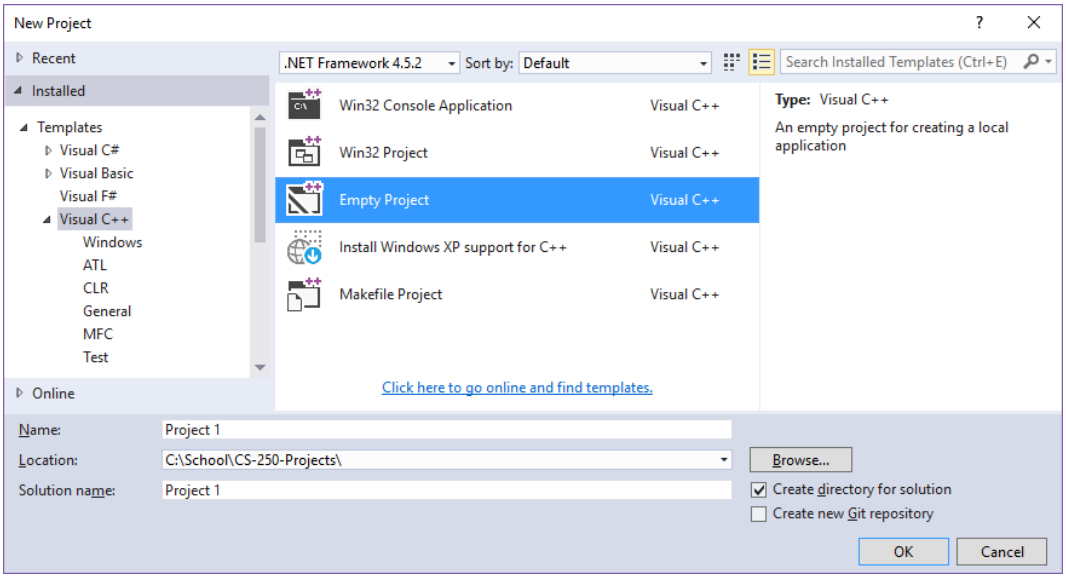
In Visual Studio

When you first open up Visual Studio, it will have a welcome page. You can select New Project from here, or from File > New > Project...



On the New Project dialog window, make sure to select Visual C++ on the left side, and then Empty Project in the middle pane. (Note: You might have to install additional files to use C++)

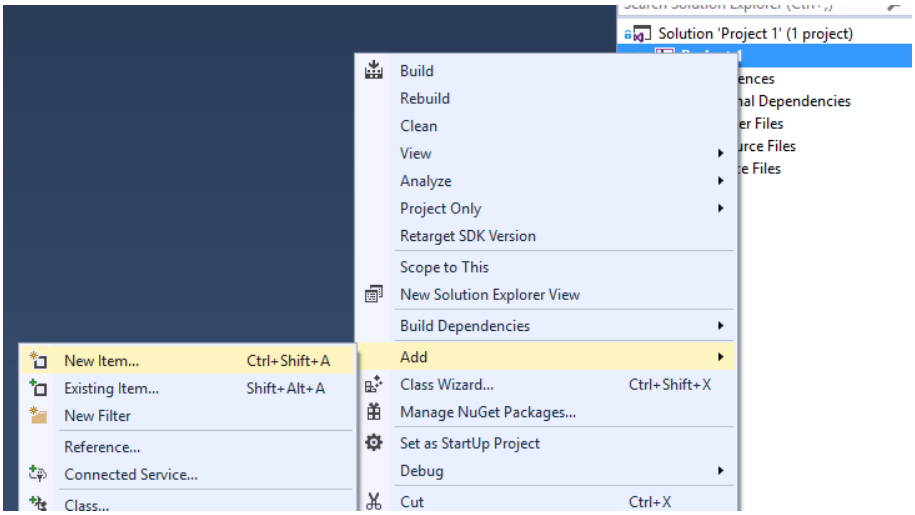
Set the project Name and Location at the bottom.



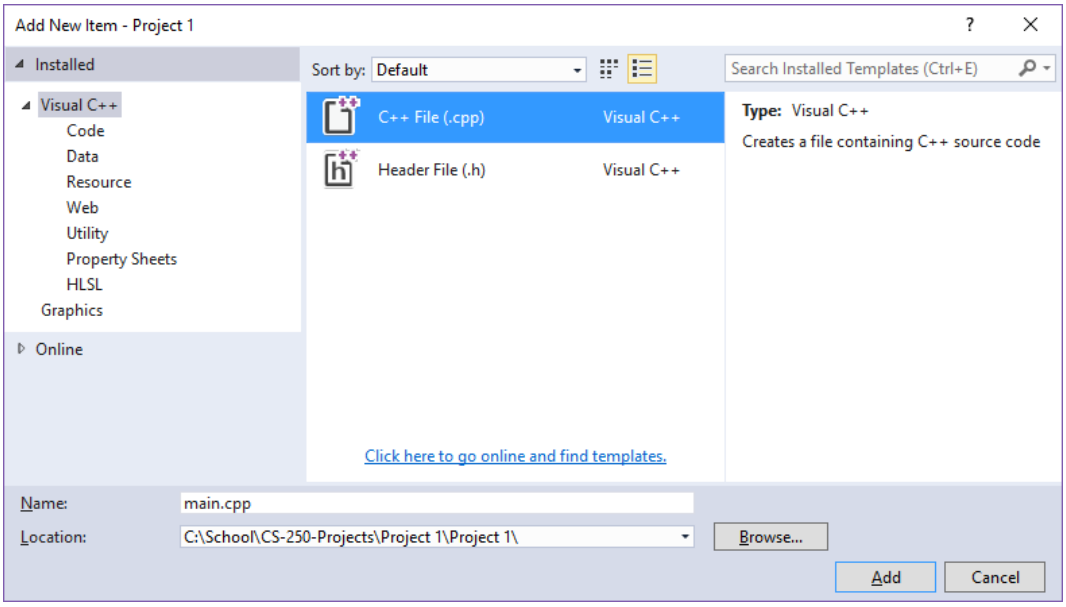
Afterwards, click on OK.

The project is created, but will be empty and contain no source files.

To add an empty file, In the Solution Explorer, right-click your project file and go to Add > New Item...



If you're creating the file to store your main() function, select the C++ File.

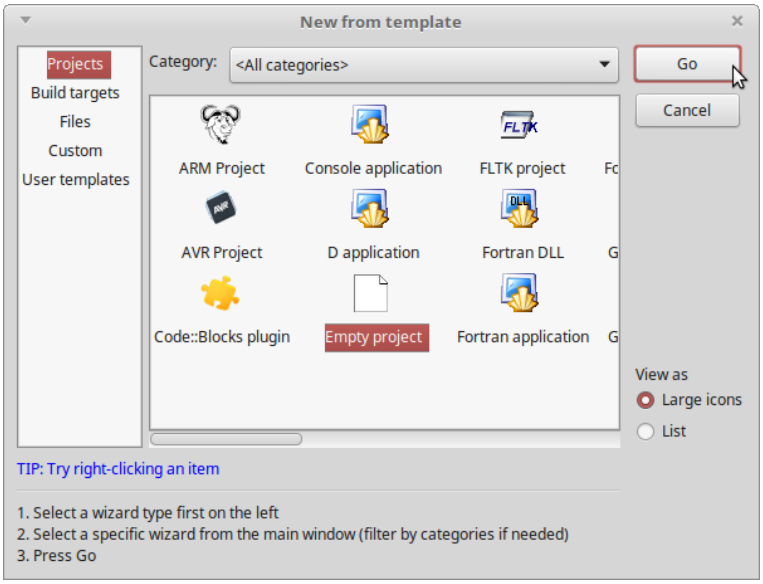


Give it a name, and click Add.

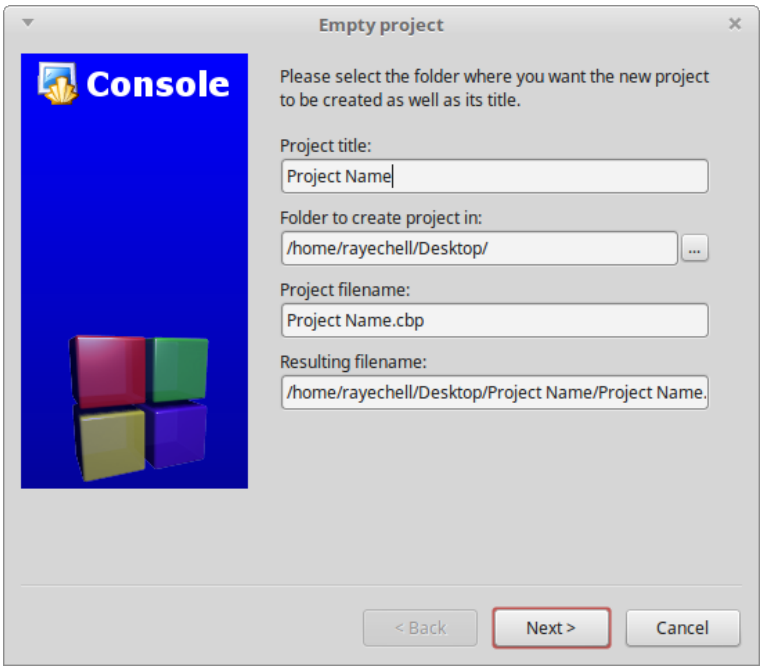
In Code::Blocks

When you first open up Code::Blocks, it will have a welcome page. You can select Create a new project from here, or from File > New > Project...

On the New from template window that pops up, select Empty project and click Go.



On the next screen...

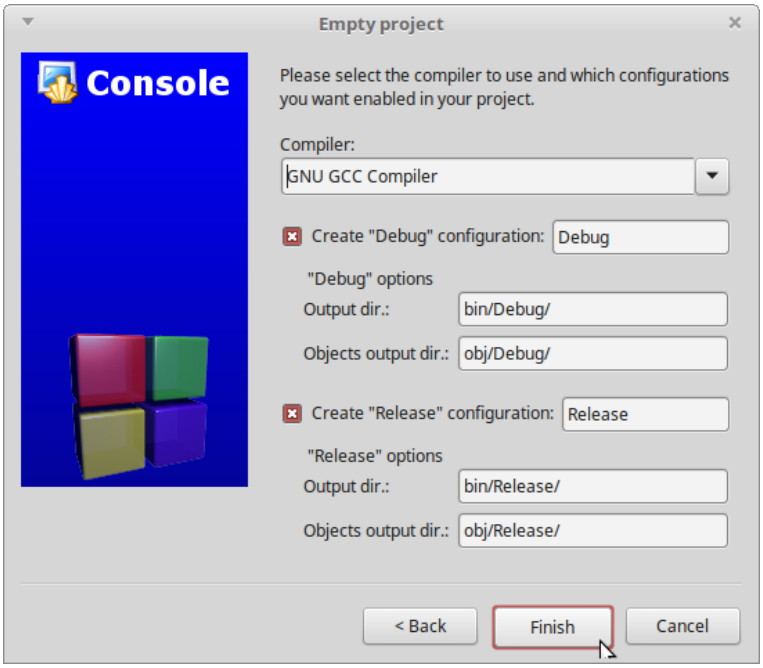


first click "..." and select a directory where you want to store your project folder.

After you have done so, give your project a title - a folder will be created for it automatically.

Then click Next.

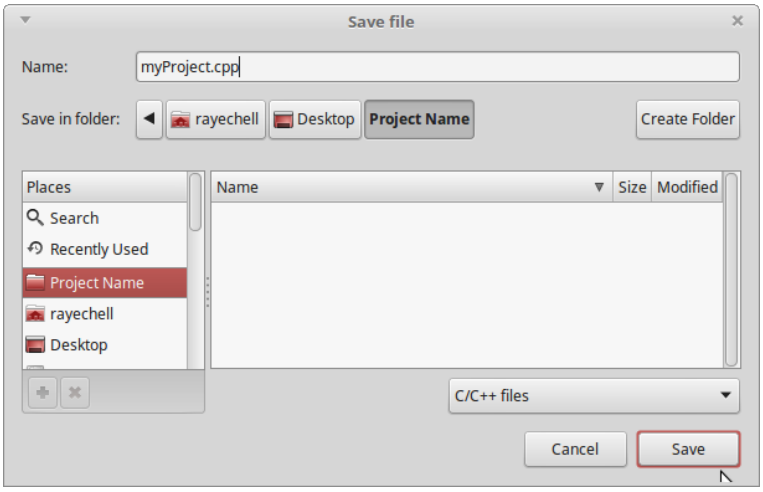
The next window asks about Debug and Release options - leave these as the defaults and click Finish.



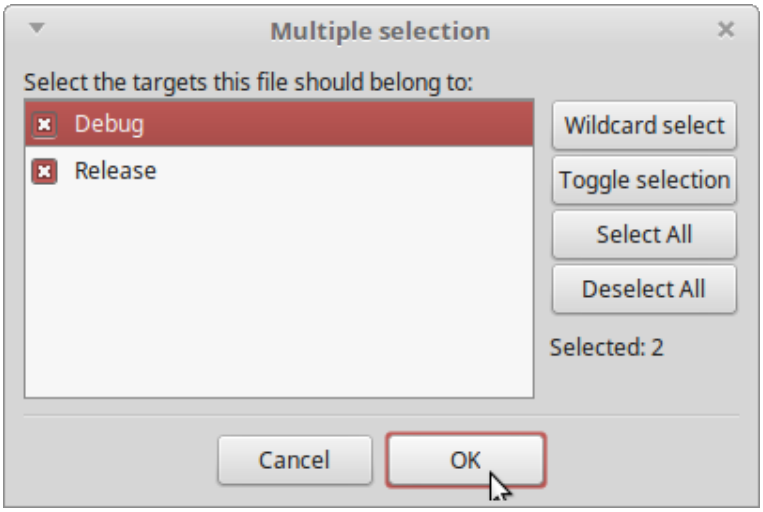
The project will be empty. To add a new file, do the following:

Go to File > New > Empty file.

It will ask, "Do you want to add this new file in the active project (has to be saved first)?" Click on "yes" and save your source file. Make sure the filename ends with ".cpp". Click "Save" once done making the file.



Another window pops up asking about debug and release. Leave these both checked and click Ok.



Creating files

For this project, you will need to make two files: A .cpp file to store your program code, and a header file, "Bag.hpp". Your .cpp file can have any name, such as "project1.cpp", "main.cpp", etc.

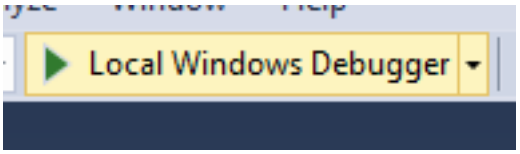
Why .hpp?
Many C++ programmers will name their header files with the .h extension, but some use .hpp. A common reason for this is if you're writing code FOR C++, not for C AND C++ together. Some libraries were written in C but can be used in both languages, while libraries written IN C++ and FOR C++ may opt to use ".hpp" as their header extensions.

Starter code

In your .cpp source file, let's start out with some basic code. Type out the following:

```
1  #include <iostream>
2  #include <string>
3  #include "Bag.hpp"
4
5  using namespace std;
6
7  int main()
8  {
9      cout << "Hello, world!" << endl;
10
11     // Don't let the program stop immediately
12     cin.ignore();
13     cin.get();
14
15     return 0;
16 }
```

Visual Studio: To build and run in Visual Studio, click on the "Local Windows Debugger" button or hit F5.



It may ask if you want to build first before you run. Click on Yes – You will always want to click on Yes here.

If the build fails, it will ask if you still want to run - say NO. Otherwise, if the build was successful, it will run your program.

Always run with the debugger
In the past, you might have gone through the dropdown menu and started your program *without* debugging so that the program wouldn't exit immediately. **DON'T DO THIS.** The debugger gives you invaluable tools for fidning errors and dealing with crashes.

Code::Blocks: To build and run in Code::Blocks, click on the "Build and run" button, or hit F9.

Make sure the program builds and runs before continuing.

PART 3

Creating the Bag class

For now, we will be working in the Bag.hpp file. First, we need to add the preprocessor directives to make sure no issues occur should the .hpp file be included from multiple source files.

```
1 #ifndef _BAG_HPP
2 #define _BAG_HPP
3
4 #endif
```

Bag.hpp

The labels here, `_BAG_HPP`, are up to you, but they should be something unique enough to cut down on the likelihood that you'd have naming conflicts.

Next, we will create the templated class declaration:

```
1 template <typename T>
2 class Bag
3 {
4 private:
5 public:
6 };
```

Templates allow us to create container classes that can store *any* data type. Instead of writing separate Bags for `int`, `string`, `float`, etc. data types, we use the type-name `T` as a placeholder.

Variable and function declarations

The Bag class will have the following private member variables:

Name	Data type	Description
`m_itemCount`	integer	Keeps track of the amount of items inserted in the array.
`ARRAY_SIZE`	const integer	Keeps track of the size of the array.

And it will have the following private member methods:

Function header	Description
`void ShiftRight( int atIndex )`	Moves everything *atIndex* and afterwards to the right by 1 spot.
`void ShiftLeft( int atIndex )`	Moves everything after *atIndex* to the left by 1 spot.

And the following public member methods:

Function header	Description
`Bag()`	The constructor, which initializes member variables.
`~Bag()`	The destructor.
`int Size() const`	Returns the amount of items store in the array.
`bool IsEmpty() const`	Returns whether the array is empty.
`bool IsFull() const`	Returns whether the array is full.
`bool Push( const T& newItem )`	Adds a new item to the end of the sequence of items.
`bool Pop()`	Removes the last item in the sequence.

Function header	Description
`void Clear()`	Clears out the list.
`int GetCountOf( const T& item ) const`	Returns the amount of instances of the given item in the array.
`bool Contains( const T& item ) const`	Returns whether the item is contained in the array.
`bool Remove( const T& item )`	Remove all items that match.
`bool Remove( int atIndex )`	Remove the item at a specific index.
`bool Insert( int atIndex, const T& item )`	Insert an item in the middle of other items.
`T& Get( int atIndex )`	Get an item in the array at a specific index
`T& GetFront()`	Get the first item in the array.
`T& GetBack()`	Get the last item in the array.

We will implement each of these one-at-a-time, and write tests to check the functionality.

Constructor and destuctor

Implementing Push

Implementing Size and testing

Implementing IsEmpty and IsFull and testing

Implementing Get, GetFront, and GetBack and testing

Implementing Contains and testing

Implementing GetCountOf and testing

Implementing Clear and testing

Implementing Insert and ShiftRight and testing

Implementing Pop and testing

Implementing Remove and ShiftLeft and testing

PART 4

Example output

PART 5

Grading breakdown