Introduction

Hello class. I just wanted to give you guys something to work on over the summer. The more I thought about this project, the more I realized I had to find a balance over when to release it. I did not wish to release this project in the middle of lab 7 because it would create a lot of confusion and I did not want to take time away from students who are busy working on their labs and other classes. It's meant to be an expansion of your knowledge. I know my personal problem when trying to come up with practice projects for a language is "What should I write?" It's hard to come up with good applications that you can easily divide into tasks, streamline your work, and check to see if you are right when you are done. This project is very simple and it helped me out in my earlier years. Plus I feel like it's really fun to be able to make a game that you can show to your family, friends, and potential employers. It is meant to be very straightforward and dip your feet into the C++ standard library as well as many C++ concepts. I suggest you expand on it once you have finished the basic case.

Disclaimer: I did **NOT** write this code. It originally came from a book I bought in 7th grade Beginning C++ Game Programming by Michael Dawson. I have added comments in a lot of the header files to teach you good programming skills since I won't physically be with you and nobody will be there to review your code.

My suggestion to you is to try to write everything on your own. Don't ask for help, don't try to look for help unless you are really stuck. The biggest problem I noticed from students during the semester is the inability to read compiler errors. Try to understand what they mean and why they occur. The more you code, the more times you see the errors, the more you figure out how to solve them on your own. For your whole career you'll find errors that you have never seen before in your life. A good programmer still asks for help, but they still try to find a solution on their own first.

After you've gotten stuck and you can't figure out what to do you will need to learn the most valuable skill a programmer can learn: how to google your problem. Reading some piazza posts from both CS 182 and CS240, I've noticed a lot of people have some distorted views about what working in the industry is like. For starters, no university can fully prepare you for your job. The reality is, companies are working on the latest and greatest stuff and it is far too specific for a university to teach it. Classes will teach you the very general concepts so that you can (on your own) figure out the rest and the more specific details. You will need to research topics, learn new tools, and learn new languages on the fly. "I have never seen it before" is not an excuse. If you wish to move up in the world of computer science, you must be able to learn quickly and adapt. That's why I find it crucial that students learn the ability to quickly and efficiently find the solution to their problem. You will understand this idea more as you get internships and have to work on real problems. This may sound incredibly difficult or stressful at first but trust me, the older you get and the more internships you have, the easier this becomes.

For your development environment feel free to use whatever editor you want. I would still suggest not using an IDE. I asked for another Teaching Assistant's experience on the matter: "From my industry experience (Facebook, DuPont Pioneer, Qualcomm), practically all C/C++ development on *nix systems was done with vim or a similar text editor. IDE like functionality was added via plugin (for basic autocompletion, ability to compile from within the editor, etc)." - Jason. I know it's all personal preference but I will keep giving my opinion on getting used to the correct environment to land you a job.

Personally at Purdue, I ended up using Sublime Text. It is a wonderful text editor that gives you some good abilities (quick scrolling, changing tabs to spaces, selecting the amount of spaces your tabs use up, tabbing an entire block at once, find, replace, and much more). If you wish to use this program, you may find it here: https://www.sublimetext.com/. I recommend a text editor because you still want to be able to write everything on your own without being told a line is wrong. You **want** compiler errors so you can learn from your mistakes. After seeing the same errors over and over again you will learn what it takes to fix them and you will associate error message with solutions.

If you wish to build this project as a Windows application or do anything that involves graphics, at that point I would highly suggest using an IDE. For Windows development, I would suggest Visual Studio. Look up tutorials online on how to make applications and how to make graphics. This topic is far too advanced for many of you so I won't go over it. For all of you who don't know, Purdue offers **tons** of free programs through MSDN. http://www.cs.purdue.edu/msdn You should most definitely take advantage of this program while you are a student. You can get free copies of Windows and Visual Studio here.

Source Control

What is source control? Source control is a source file management system which allows you to keep track of all your changes to your source files. Before you even start coding, get yourself a private repository. This is the most important advice I can give to any student starting off in Computer Science. For those of you who already have source control or know git, please skip ahead to the next section.

Where do I get source control?

There are many websites you can use. I personally prefer GitHub because I've used it at work since many companies can buy a private repository hub for their small company or teams. This is their website https://github.com/ . As students, you are allowed to get 5 free repositories. https://education.github.com/discount_requests/new this will let you fill out an application as a student to get your free private repos. I like the interface and it is easy to add collaborators to your projects. Another common and popular host for remote repositories is Bitbucket. This is

their website www.bitbucket.org they have an infinite amount of private repos, which is perfect for students.

The keywords you are looking for are “private repository”. Private means other people in the internet cannot see your code. Using a public repository means other students at Purdue could find your code and you can easily get reported for academic dishonesty. My personal workflow while you are here at Purdue is this:

1. At the beginning of the year, clear any old private repos.
2. Create a private repository for all your CS classes. (Github offers 5 private repos at any time).
3. Add all of your project for a given class to this private repo.
4. At the end of the semester, download your repo as a zip file and place it in your local computer for storage. You’ll end up reusing code for later classes. For example, as a graduate student, I used code from my CS 252 class.
5. Never lose your code again and enjoy life. Because it’s on the internet you can view and edit your code anywhere in the world. Especially if you have to travel home for a weekend.

I would recommend setting up your SSH keys in order to not have to login from data.cs.purdue.edu onto GitHub, as an example. SSH keys are an authentication mechanism used by many applications in the real world. For a quick tutorial on how to generate and add your SSH keys to GitHub follow these instructions: https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/

For anybody who has never used git before, I have added a whole tutorial with this tarball, or just in case you wanted to know how git is used in the industry. It is an incredibly powerful tool. Source control is one of the most important tools you must use as a student in order to always have a backup of your code.

How to Start

First off, do **not** look into the solution folder. Obviously I wanted everybody to have a solution so they can see two things: 1) If you ever get stuck you have a way of looking at the solution for a given class or method. 2) Some good programming practices.

You get good at coding by coding a lot. You get a good at writing good code by having your code be reviewed by people who know more than you. Since you are doing this on your own, you’ll need some sort of reference to find out how people code in C++. Once you have finished everything compare your code to this code and ask yourself “should I have written my code any

better?” There are techniques you are not used to writing yet because you don’t write C++ code. I’ve tried adding hints about good programming practices in the header files and cpp files.

For the first few times, avoid using a makefile. Try to compile the code with g++ directly from f memory of the commands. You want to memorize the g++ lines and be able to do it for any task. Do you remember what flags to use? Were you able to compile right away with no errors? If you didn’t, continue writing the compilation commands by hand every time. Don’t press the up key in the terminal to use the previous command you typed. Once you are no longer making mistakes compiling manually, create a makefile. This is a learning process and you are only going to get good if you are actually trying to learn.

Compare your makefile with the provided one once you are done, were you able to get the same makefile? Did you have all the parts of a makefile? Were there some lines you forgot? These are all things you should be asking yourself. Once you are done with the whole project, compare your .cpp files with the solution .cpp files. I have left you messages inside some of them to show you how to code. Could you have written your code with fewer lines? How so? How could you improve? This is how you get better at coding practices. You need to emulate these techniques on your own. Make sure you identify what common mistakes you are making so you can improve.

Then before you start coding your project. I suggest you look at the code inside the lesson directory and study it. They will give you an example of some of the methods of the Standard library and the Standard Template Library and how to use them. They will come in handy for the actual tasks.

The Standard Library

There is some merit in forcing students to build the basic data structures from scratch at least once. Knowing how these data structures work is crucial for software engineers. In any job interview, these will be seen as the easy questions. Not being able to describe how any basic data structure works or being able to write them on your own will usually result in an instant failure. Now that you do know how to write a lot of these data structures though, it’s time to stop reinventing the wheel and code like real programmers.

C++ includes some special Standard Template Library (STL) classes called containers. Some of these names may sound familiar to you: deque (double ended queue), list (linear list), map (normally implemented as a red-black tree with unique keys), multimap (same as map but duplicate keys are allowed), multiset (normally implemented as a red-black tree. Only uses keys but not values), priority_queue, queue, set (hashset), stack, and a vector (an arraylist). These containers are important because they are templates and will work for many different types **efficiently.** This is code used by lots of programmers that has been refined over the years and fixed to have the least amount of mistakes. It is much better to learn how to use existing code

from libraries like the STL than to create something from scratch from the start. (**Note:** all of these containers belong to the standard library, meaning they would need to be called by std::<class>. **Do not** use “using namespace std”. It is time to learn things right because once you get into a bad habit it’s hard to undo. You will not be able to use that line of code in the industry and it’s better to get familiar with what real programming looks like.

You will learn about the efficiency of each container in CS 251. You’ll learn what methods are inefficient and which are efficient, and more importantly, why they are or are not efficient. Learning this is crucial as it makes you a better programmer. It makes you think about solving problems efficiently instead of simply coding for “the right answer.” This is the transition from a freshman into a junior. By the time you are a senior you most definitely need to know about efficiency in all your solutions, as getting a job (after your junior year but before your senior year) will depend on coming up with the optimal solution during interviews.

Vectors

Since you are unlikely to know about containers right now, I’ll go over vectors and the std::vector class for you (because you will really need them for this task). For the other containers in the standard template library, you should research their methods, what they do, how they are called, how they are used, and so on by yourself. (**Note:** For interviews, the hashmap/std::unordered_map is probably the most important of all data structures because of it’s ability for constant time lookups and insertions).

What **is** a vector? It’s a resizeable dynamically growing array. It has the benefit of quick access to any element just like an array. It performs poorly when you try to insert something that is not at the end and when you try to remove something that is not at the end. Why is that? Because just like in an array, we like to have no gaps between our data. Hence you would need to shift all of our data by one for each of these operations. If we are adding or removing at the end, no shifting will be required and the element can be added instantly.

Now that we know these things let’s find a resource with all the information that we need about the std::vector class which implements a vector.
http://www.cplusplus.com/reference/vector/vector/ this website seems to have exactly the class that we are looking for. On top of that, we can easily click on links to any of the methods the std::vector class offers to find examples and more detailed information on a given method. The three methods we have discussed so far are called with operator[] (for getting a value just like in an array), push_back() (to insert at the end of our array), and pop_back() (for removing a value at the end of our array). Now you should research the other functions (but ignore the iterators for now) to see if you understand how to call the methods. Look up the lesson directory to learn more about vectors and how useful they can be.

Iterators

One of the other benefits of these standard library containers is that they come with these wonderful iterators. Iterators are objects that contain the state necessary to track where in a container you are currently iterating. Iterators allow you to easily traverse a container.
Without getting too far into the weeds, there are several different categories of iterators that give you different capabilities. Each container in the STL offers a container specific iterator class with specific capabilities, such as random access by index, sequential access, read-only access vs. read-write access, etc. To get an instance of an iterator for a given container instance, you call begin() on that container instance. The iterator that the begin() method returns points to the first element in the container and is then used to access elements of the container. To access an element of the container, you use the dereference operator on the iterator ('*') which returns to you a reference to the element inside the container that the iterator current points to. You can use the post-increment operator ('++') on the iterator to advance it to point to the next element in the container. You can continue to use the '++' operator to move the iterator forward until the iterator is equal to the value of the return value of the end() method called on the same container. The value end() returns is a special iterator value that when compared '==' with another indicates the other iterator has reached the end of the container.

Iterators may seem simple, but they are used by other standard library functions that could really speed up your life. A major premise of the STL is that you can implement algorithms that take iterators with specific properties (read-write, random access, for example) to ANY container type that supports those properties on it's iterators, and then be able to implement any algorithm on the container (sorting, find the maximum, find the minimum, take the average, etc). The idea is to separate the details of a container that stores elements such as a vector, from the details of the function that implements an algorithm. One such function would be the sort() function that is declared in the <algorithm> header. This important header file that you may wish to include in your projects can be included by using '#include <algorithm>' in your .cpp files. By including <algorithm>, you will have an efficient sorting algorithm which works with iterators. A good source of information with examples exists right here: http://www.cplusplus.com/reference/algorithm/sort/**.** Now you know how to sort a vector. You can look in the lesson directory to see how you could have printed the sorted IDs of Employees in lab 7. You will also need to know how to traverse a vector with an iterator (and how to get an iterator in the first place). This lesson plan may help you understand. (TODO, is the preceding sentence a fragment?)

Sometimes you wish to traverse an object backwards. In the case of a vector, imagine going from the end all the way to the front. For that reason, some containers offer an object called a reverse_iterator. These reverse_iterators work the same as a normal iterator but allow you to traverse the list in the opposite direction. Again, please look in the lesson directory for an example.

Blackjack Rules

For the sake of this project, we will be simplifying the rules of blackjack. For starters were are only going to be using 52 cards or one standard deck. These have the values ace, 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, and the King. The suits do not matter in blackjack.

In blackjack, the objective of the game is come as close as possible to 21 points without going over. Points are based on the cards you have. An ace is worth either 1 point or 11 points. The numbered cards [2, 10] are worth exactly as much as their number. Finally the face cards (Jack, Queen, and King) are also worth 10 points. As you can see, the easiest way to win the game would be to have an Ace along with either a 10, a Jack, a Queen, or a King.

If the point value of a player's hand goes over the value 21, he or she "busts" and loses. If a player's point values matches the same as the dealer her or she "pushes" and neither the house nor the player win. Any player whose points are higher than the house but below 21 wins. Any player whose points are lower than the house loses.

The house begins by giving each player 2 cards (all face up) and then one face up card and one face down card for itself. Then each player will be continually asked if they wish to take a card (hit) or stay with the point they have. Once every player has either busted or decided to stay, the house will flip their face down card and either hit or stay depending on how many points it has.

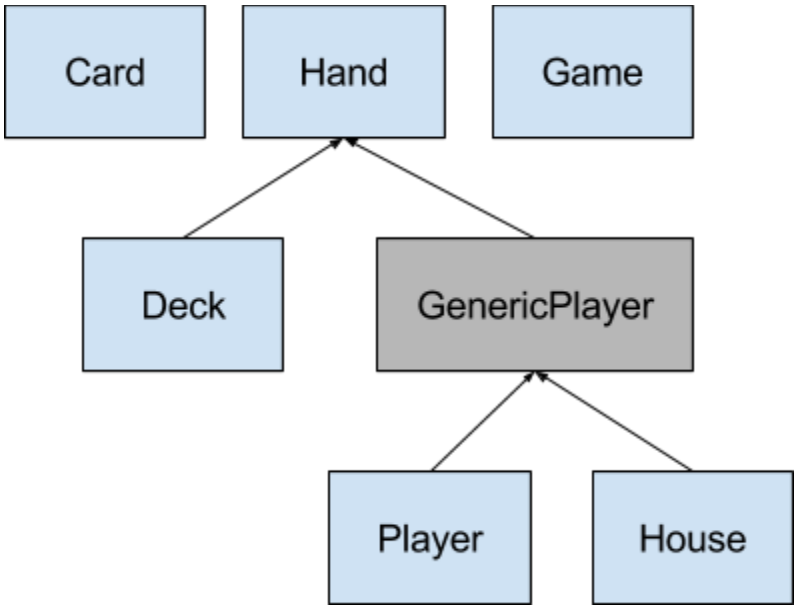
To help you understand how the game works, I have included an executable file so you may explore the rules on your own without seeing the compilation line (so you can try to figure it out on your own).
Note: Executables usually only work on the machines they were compiled because the format of the executable produced by the compiler is specific to both the architecture of the CPU and the operating system. So an executable compiled on a Linux x86-64 machine (like the ones in the labs) will *not* work on a Mac x68-64 machine. The executable I've included was compiled on data.cs.purdue.edu. If you are working on your local machine this executable may not work (sorry).

Tasks

Each the classes are broken up in the following way:

Class	Base Class	Description
Card	None	A Blackjack playing card
Hand	None	A blackjack hand. A collection of Card objects
Deck	Hand	A Blackjack Deck. Since a Hand is just a collection of Cards, a Deck can be thought of in the same way but with extra methods.
GenericPlayer	Hand	A generic Blackjack player. This is an abstract class which holds all the common elements between a human player and a computer player
Player	GenericPlayer	A human Blackjack player
House	GenericPlayer	A computer Blackjack player (the House).
Game	None	A Blackjack game.

To put it into a picture. The classes look like this:



Let's get this out of the way first: the **entire** project is extremely simple to code. Expect each file to be between 20 lines of code and 60 (except for the Game class). This is only meant to be a stepping stone into programming larger projects and it gives you an idea of how to make your own projects as well as giving you programming tips which you might not know.

I have put all of the instructions in the header files and cpp files. Each function has a description of what it is supposed to do (in some cases this should be very obvious). Every header file has a guard to prevent it from being included twice. They also draw from a common header that contains #include statements for commonly used header files. Look into common.h to see what tools you have at your disposal.

My suggested ordering is:
Card, Hand, Deck, GenericPlayer, Player, House, Game, Main.

You finished!

So you have finished the project, you've checked the solution file, you are satisfied with your work, but you still want to learn more. What should you do now?

Check for memory leaks

One of the easiest things to do first is check that you are not creating memory leaks in your code. Don't forget C++ allocates a lot of memory for you automatically. Being able to fix your own memory leaks will make you a better programmer. Being conscious of memory leaks will also make you a more cautious programmer. Unfortunately I do not have time to explain every memory leak in C++. The CS 390 class with Gustavo that covers C++ will probably not go into details either, but it may be enough to explain why some memory leaks do happen (shallow copy vs deep copy).

Project Expansions

Other Card Games

Now that you have this project for blackjack, think of expansions to this game. A good second project would be to change the game from blackjack to Poker. Maybe you wish to create a different card game altogether. Try creating the card game Uno. What about Artificial Intelligence for a computer player? For our simple game we had very easy artificial intelligence. The computer will choose to hit whenever they have less than 16 points. Maybe you wish to understand card counting and probabilities. You are free to add whatever you want and this could start you off into interesting topics in computer science.

Real Blackjack

Blackjack is a game played in casinos. So far we very straightforward rules to playing the game. Casinos usually have more than one deck of cards to prevent people from counting the cards (sometimes thought of a cheating). There is also a rule for when a player gets that same card twice. A player has the option to split their hand into two. Now a player can win twice, win in one, or lose on both hands. On top of that, each player is given chips (or money) they can use to play the game. You can introduce chips into the game and keep track of winnings. Perhaps setting up a high score screen for your blackjack game. Can you add the code to split a hand? Can you add multiple decks and create better Artificial Intelligence for the house?

Start by creating a menu for your game. A player could try pressing the number keys to perform different options. Create a default option in case the player tries an invalid option. For a high score screen, learn how to read and write to a file in C++ to display names. Add chips into the game. Allow players to begin each round by betting a different amount of chips. Kick players out of the game when their chip count reaches 0.

Now that we have this title screen which allows us to play blackjack, what if we had other card-based games as available possibilities? You will probably want to split up Game.cpp into two files: Game.cpp (a simple main function to keep track of your title screen), and Blackjack.cpp where you keep track of all the blackjack logic. Then create new files like Poker.cpp and Uno.cpp where the game logic for those games can be written. Have your Game.cpp create an object based on the game type to play it! That would be a great example of using inheritance and polymorphism with a Game class that you subclass for each type of game!

New Games!

Another (but much bigger) step would be to create a new project or game which is not card-based. For this you would need to come up with a set of predefined rules for the game. Can you come up with the rules for chess? Don’t forget this is object oriented programing. Can you think of what classes you would need to create and why? What methods would these classes have? In real projects, the design takes up the most time. You will understand this concept more if you take CS 307 Software Engineering. Jumping into coding without planning ahead is an incredibly bad idea and leads to failure. It would be a great learning experience to come up with a new project on your own and spend 80% of your time designing and 20% of your time coding. Show your projects to your friends and family! Ask for suggestions and improvements including things they would want to see.

Graphics

For a really tough project create some graphics for your game. For this, I would **highly** suggest using an IDE and tutorials on app programming. C++ is one of the best languages for applications so there will be many tutorials and youtube videos. It is truly a unique experience, I'll say that much :)

Thank you for everything

Thank you for taking the time to read this. I know there were tons of comments and files. I hope this helped at least some of you enough and that you found this project to be worthwhile. Good luck at Purdue and with all your future classes. Continue improving and continue learning! I hope to see you one day at whatever company we may both end up at.

I would also like to thank Jason Rahman for being such a wonderful friend, helping me correct grammar errors and terminology, sharing his personal advice on some of the topics, and giving me some input on what to include and exclude in this incredibly long and wordy handout.