





**Resources**

[Resources](https://faytechcc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_33824_1&content_id=_5842758_1)

**Course Playlist**

The CSC 234 YouTube playlist is now available.

2020FA CSC234: <https://www.youtube.com/playlist?list=PLKM4GUn71woe__DR5JjXWjNcD7qgOxkx_> (opens in new window)

I've started with an introduction that discusses the merits of repl.it versus Code::Blocks. If you've already found a preferred method to compile and run C++ programs on your home machine, you can skip this, but hopefully some people will find it useful. I'll add future videos as relevant.

**Reference Docs**

The following file is the Gaddis "C++ Language Companion". You may find this file useful for reference.  [C++ Language Companion.pdf](https://faytechcc.blackboard.com/bbcswebdav/pid-5842761-dt-content-rid-54063186_1/xid-54063186_1) [C++ Language Companion.pdf - Alternative Formats](https://faytechcc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_33824_1&content_id=_5842758_1)

**Notes on "Adventure Call"**

We're starting with a Choose-Your-Own-Adventure style (see <https://en.wikipedia.org/wiki/Choose_Your_Own_Adventure>) game, where each choice leads to a different "node", a "node" in this case just being a new function in which we state the situation, ask for a choice, and then branch to new nodes.

This format is pretty easy to work with, but has some major limitations -- every single eventuality in the game has to be covered by a separate function.

Where we're headed as we push into Object Oriented programming is a version 2 of this strucure that uses some underlying **data structure** for the series of rooms the player can walk through (as in the 9:05 example). With this structure a game's size is limited by the size of the underlying room data, rather than the number of functions.

### Topic: "Divide And Conquer"

[Topic: "Divide And Conquer"](https://faytechcc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_33824_1&content_id=_5842762_1)

**Abstraction**and **Divide and conquer**,

Think of a large, complex bureaucracy -- thousands of forms may get passed around each day, but each individual worker does one thing at a time: "Get form from inbox, process the form, put form in outbox". Programs are structured in a similar way, with each piece doing one specific job and having defined input and output.

**if, switch, and debugging "curly brace hell"**

**if**and **switch** do the same thing -- choose one branch out of multiple options. **if**is the right choice for two branches (do a thing, or don't), which **switch** is the way to go if you're choosing, say, one option out of three, or five, or ten.

Starting out, most of the time you'll use **if,**and cases where **switch** make more sense will be obvious.

This is typically when students start cursing misplaced curly braces and semicolons. Branching code is one place where it's easy to forget to close your braces. (Every time you type a { in code, there has to be a matching } later on. Where you place these is vital.)

If you've ever suddenly been totally lost while following directions, and not sure how many steps back you got off the path, that's what a compiler is dealing with when there's an issue with braces. This means that the error messages (and error line numbers) are not particularly helpful. When in doubt, go back and check your braces. In most editors, when you select a {, the editor will highlight the matching } (or vice versa).

**Functions**

Functions are the building block of complex programs. Up until now, your program has contained one function, **main()**. The main() function is required, so that the compiler knows where your program starts. Generally it runs once, and when it's finished, the program is over.

Other functions may be called many times during a program run. In most programs you'll both use predefined functions, and functions that you define yourself. 

Try not to get hung up on the specific **square()** example -- I know just typing "x \* x" would be faster. The point is to abstract away some detail so that the person who calls **square()**doesn't need to know how it works internally.

**Void vs. Value-Returning functions**

A void function ends without a return (it's implied you return when you hit the closing curly brace). A value-returning function ends with return(some\_value), where some\_value typically comes from a variable.

Listing 5-12 in the book uses arguments passed into main(). This is handy if you write, say, a batch file or shell script that uses a C++ program to do some of its work, but our programs don't need this functionality since we get all the input we need inside the program itself.

Remember that returning zero is not the same as returning void! Void functions don't have a return statement at all.

Sometimes we "cheat" by having a value-returning function whose returned value we ignore. main() is an example -- we return zero if the program was successful, non-zero if it failed. However, we don't ever actually look at that value in this class. If you were to write a script (a batch file or bash script) that used a C++ program you wrote, it might use that return value.

**Parameters / Arguments**

The book talks about parameters -- some other sources will sometimes talk about arguments. Both describe inputs to a function.

I use the terms **formal parameters** and **actual parameters**.

Formal parameters are used in the function definition, to "formally define" what inputs the function takes. Actual parameters are used when you call a function, to bind those formal parameters to real values.

For example, you may write the function square(int value) , where value is helping to define the structure of the function -- that it takes one integer.

If you call the function in some code that is doing geometry, you might call it like this: area = square(length); Here length contains some integer, and that number is the actual parameter.

### Adventure Call

[Adventure Call](https://faytechcc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_33824_1&content_id=_5842762_1)

You can find my partially completed Adventure Call program here:

<https://github.com/norrisaftcc/csc134> (opens in new window)

This version uses global variables to pass information between functions. Using global variables is generally a bad idea -- but it's a shortcut we'll use for a little bit. A better design would use a C++ class to hold our state information, and that's something we'll do later.

The particular reason a global variable is used in this version is to require a key from the gazebo in order to enter the shack. The variable is set in one function, and checked in another. Can you think of a better way to handle this issue rather than using a global?

### About Interactive Fiction

[About Interactive Fiction](https://faytechcc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_33824_1&content_id=_5842762_1)

About IF (Interactive Fiction):

For the function-based project we're working on at the moment, the idea of "Choose Your Own Adventures" or "Text Adventures" is a good starting point. This type of game is often called "interactive fiction", and there's a small dedicated community still making these kinds of games.

Beginner's Guides to IF:

<http://www.ifwiki.org/index.php/Starters>

<http://brasslantern.org/beginners/>

**Sample IF Games**

You can find some games at: <http://www.web-adventures.org/> and <http://textadventures.co.uk/>. "Galatea" is pretty experimental, but easy enough to play. "Zork I" is old-school, and pretty difficult.

I recommend you try either "9:05" or "Photopia" by Adam Cadre (you'll have to search them yourselves, as the links are inexplicably blocked by the school web filter, but they're easy to find.) Both are fairly short introductions.

If rather than playing 9:05 you'd rather read a transcript, you can find one here: http://www.allthingsjacq.com/intfic\_clubfloyd\_20100903.html

**Lemmy's Adventure Call:**

(This is a series of skits about a "play-by-phone" interactive fiction game.)

https://youtu.be/d401ABH5Pj0

https://youtu.be/PzKo1JPNf4s

https://youtu.be/kkxGynvgvRU

### Game Design - Some Basic Ideas

[Game Design - Some Basic Ideas](https://faytechcc.blackboard.com/webapps/blackboard/content/listContent.jsp?course_id=_33824_1&content_id=_5842762_1)

The point of this discussion is to churn up some ideas that might be useful when coming up with ideas for your project, and possibly for future projects. If we're doing software projects, something game-related presents an interesting design space.

In other words, yeah, you can make a game right now without thinking about any of this stuff. But it might be useful later.

While pen and paper ('tabletop') role playing games are different from computer games, there's a lot of ideas that you may find useful. It might even be possible to make an 'early access' version of your game that runs with dice, paper, and a human in the role of the computer.

In fact, one tabletop RPG, Murderous Ghosts, works very much like a human-driven text adventure. One player is a person who's wandered into an abandoned building, the other is a ghost. Each turns to page 1 and makes a choice from those provided, and as they play each turns to different pages for their next move. Something like this could work well as a text adventure.

(Review of Murderous Ghosts: <https://www.dreadcentral.com/reviews/282253/murderous-ghosts-review-choose-your-own-poltergeist/>)

**What Kind of Game?**

While you could do just "straight up Blackjack", we're examining types of games that when played with paper and dice at a table are called "Role-Playing Games", and when played on a computer are called things like "RPGs" or "Immersive Sims" or "Open World". The point being, it's not just mechanics -- you're trying to put the player as a character somewhere.

**What's the Hook?**

In order to grab a player's interest, this type of game usually wants to feel like it's putting you in a **Situation**. One way that this is described is:

**Situation = Setting + Character + Conflict**

although Conflict is often implied in the other things. So, for example, "Star Wars" is a Setting, but it's a little too broad -- that could apply to all kinds of games. But "Star Wars, and you're a janitor on the Death Star" gives you a fully formed SItuation. (Both the Conflict of "You've got to clean up all these laser blasts and dead Stormtroopers" and "your office is going to blow up soon" are implied.)

("You're a janitor" games do exist -- see Viscera Cleanup for an example.)

It's possible to come up with a Situation and pair it with an existing game.

For example, "Blackjack", by itself, has no Situation. How about "You're an Old West cowboy. You're playing Blackjack, with the Devil, and the winner gets your soul. You have one shot left in your trusty six-shooter." There's Setting, Character, and Conflict, so you have SItuation.

(How could this SItuation possibly influence the gameplay? Maybe the Devil cheats. Maybe he chatters away between hands, and is either giving hints, or trying to trick you. Maybe you can use that last bullet to put an extra hole into a card, and turn a 3 into a 4, to win a tied hand.)

**Conflict, Uncertainty, Game Mechanics**

In a tabletop RPG, you generally have one of the people at the table acting as "Game Master", sort of running things, and the rest of the players playing Player Characters. With a computer, we've got a simpler setup, but the analogy might be useful.

The computer is responsible for:

* 1. Telling the player what's going on,
  2. what they can do,
  3. and what the results are.

For (1), because we're starting with text-based games, some of that "human GM" element is something we should think about, because what "graphics" the player sees are basically what they imagine when they read the text that the computer prints out.

For (2), to keep our programs a manageable size, we're not going to try to allow the human player to do anything they can possibly think of. We'll either use a **menu** to let them pick an option, or a **text parser** to let them describe what they want to do. (You can think of the text parser as a semi-hidden menu. It doesn't give them more options, it just lets them pick them differently.)

For (3), some kind of **game mechanics** will be needed to decide results.

With people at a table, there are roughly three ways to decide outcomes - Drama, Karma, and Fortune. Here's some examples.

* 1. Drama: "It would be more interesting if you didn't defeat the Big Boss yet... so you don't."
  2. Karma: "You're Level 99 with 200 Strength, you defeat the Big Boss automatically."
  3. Fortune: "You have a 20% chance of defeating the Big Boss."

In practice, these often get mixed together, usually some combination of Karma and Fortune. (For example: "You start at a 0% chance of beating the final boss. Every level you gain and every point of strength you get increase that chance by 1%.") Drama doesn't come into play as a mechanic, other than maybe a pacing mechanism (don't send level 1 enemies against a level 50 player character, because it would be boring.) Note that when RPGs use Drama with a heavy hand for things like "You have to lose this boss fight", it often irritates players.

So Karma + Fortune usually means "You have a random chance to succeed, and that chance depends on how prepared you are", with things like leveling up and getting equipment. But it could also apply to a detective game, where clues give you a better chance of finding the killer.

Here's some one-page examples of a tabletop RPG, with mechanics simple enough that you could use them pretty easily:

Lasers and Feelings - <http://onesevendesign.com/lasers_and_feelings_rpg.pdf>

Honey Heist - <https://www.docdroid.net/KJzmn5k/honey-heist-by-grant-howitt-pdf>

Here's a simple tabletop RPG system that probably doesn't adapt easily, because it includes rules for the player deciding in an open-ended way what happens next: The Pool - <http://www.1km1kt.net/wp-content/uploads/2013/07/thepoolrpg.pdf> . However, if you replaced "the player can narrate what happens next" with "the player chooses from a menu what happens next", this system would work for a CRPG type game.