How to Create a Program

Computer programs are implemented everywhere these days, from our cars to our smartphones, and in almost every job. As the world becomes more and more digital, the need for new programs will always keep increasing. If you have the next big...

Part 1 of 6:

Coming Up With an Idea

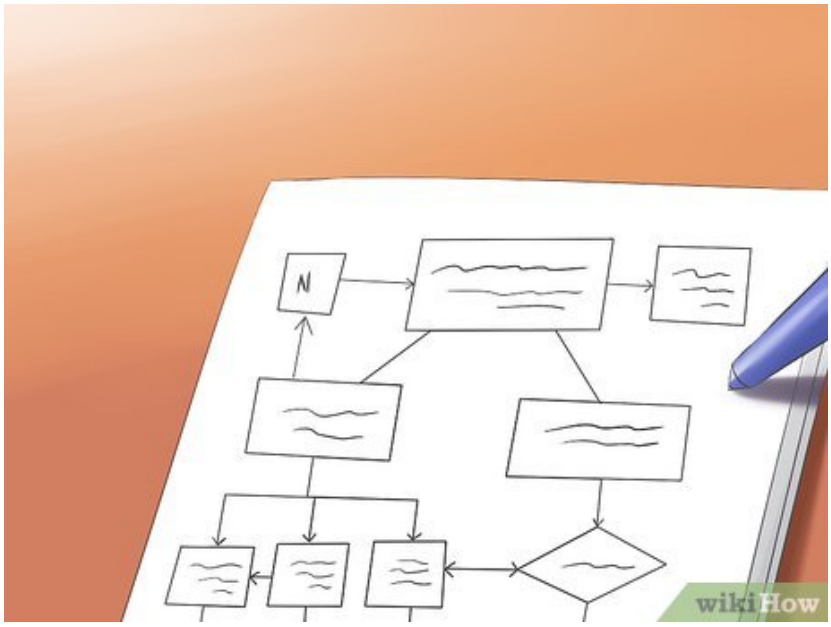
1. 

Brainstorm ideas. A good program will perform a task that makes life easier for the user. Look at the software that is currently available for the task you want to perform, and see if there are ways that the process could be easier or smoother. A successful program is one that users will find a lot of utility in.

1. Examine your daily tasks on your computer. Is there some way that you could automate a portion of those tasks with a program?
2. Write down every idea. Even if it seems silly or outlandish at the time, it could change into something useful or even brilliant.

2. 

Examine other programs. What do they do? How could they do it better? What are they missing? Answering these questions can help you come up with ideas for your own take on it.

3. 

Write a design document. This document will outline the features and what you intend to achieve with the project. Referring to the design document during the development process will help keep your project on track and focused. See this guide for details on writing the document. Writing the design document will also help you decide which programming language will work best for your project.

4. 

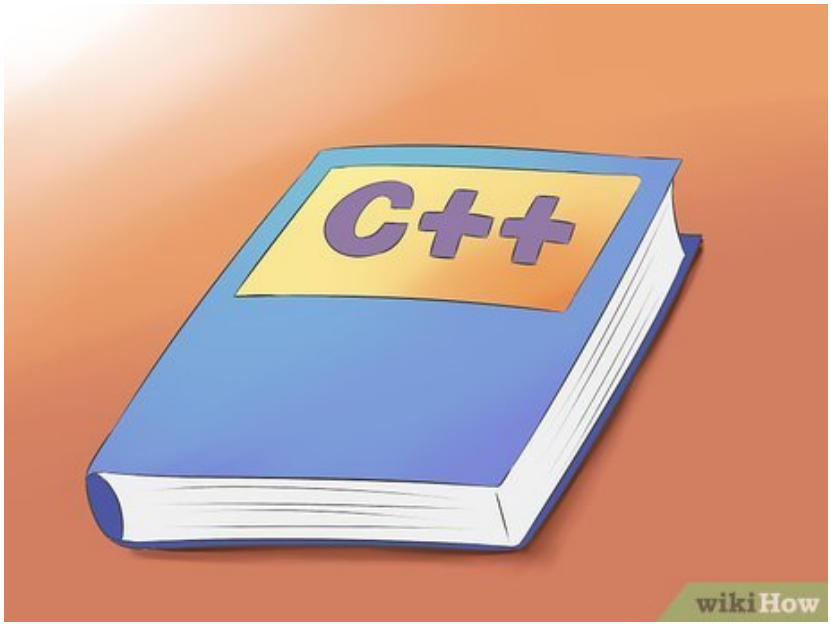
Start simple. When you are just getting started with computer programming, it will behoove you to start small and grow over time. You will learn a lot more if you set tangible goals that you can reach with a basic program. For example,

Part 2 of 6:

Learning a Language

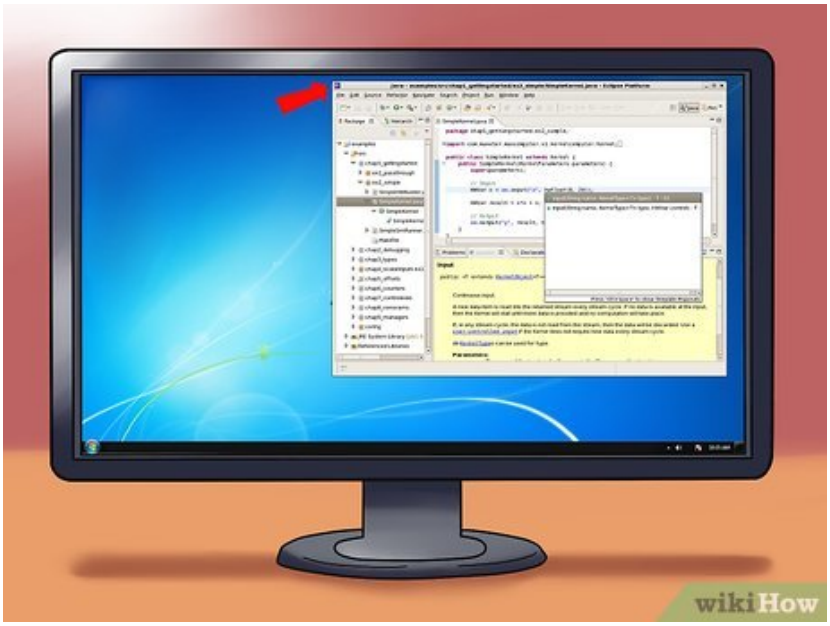
1. 

Download a good text editor. Almost all programs are written in text editors and then compiled to run on computers. While you can use programs like Notepad or TextEdit, it is highly recommended that you download a syntax-highlighting editor such as Notepad++ JEdit, or Sublime Text. This will make your code much easier to visually parse.

1. Some languages such as Visual Basic include an editor and compiler in one package.
2. 

Learn a programming language. All programs are created through coding. If you want to create your own programs, you will need to be familiar with at least one programming language. The languages you will need to learn will vary depending on the type of program you want to create. Some of the more useful and important ones include:

1. C - C is a low-level language that interacts very closely with the computer's hardware. It is one of the older programming languages that still sees widespread use.
2. C++ - The biggest drawback of C is that it is not object-oriented. This is where C++ comes in. C++ is currently the most popular programming language in the world. Programs such as Chrome, Firefox, Photoshop, and many others are all built with C++. It is also a very popular language for creating video games.
3. Java - Java is an evolution of the C++ language, and is extremely portable. Most computers, regardless of operating system, can run a Java Virtual Machine, allowing the program to be used nearly universally. It is widely used in video games and business software, and is often recommended as an essential language.
4. C# - C# is a Windows-based language and is one of the main languages used when creating Windows programs. It is closely related to Java and C++, and should be easy to learn if you're already familiar with Java. If you want to make a Windows or Windows Phone program, you'll want to take a look at this language.
5. Objective-C - This is another cousin of the C language that is specifically designed for Apple systems. If you want to make iPhone or iPad apps, this is the language for you.

3. 

Download the compiler or interpreter. For any high-level language such as C++, Java, and many others, you will need a compiler to convert your code into a format that the computer can use. There are a variety of compilers to choose from depending on the language you are using.[1]

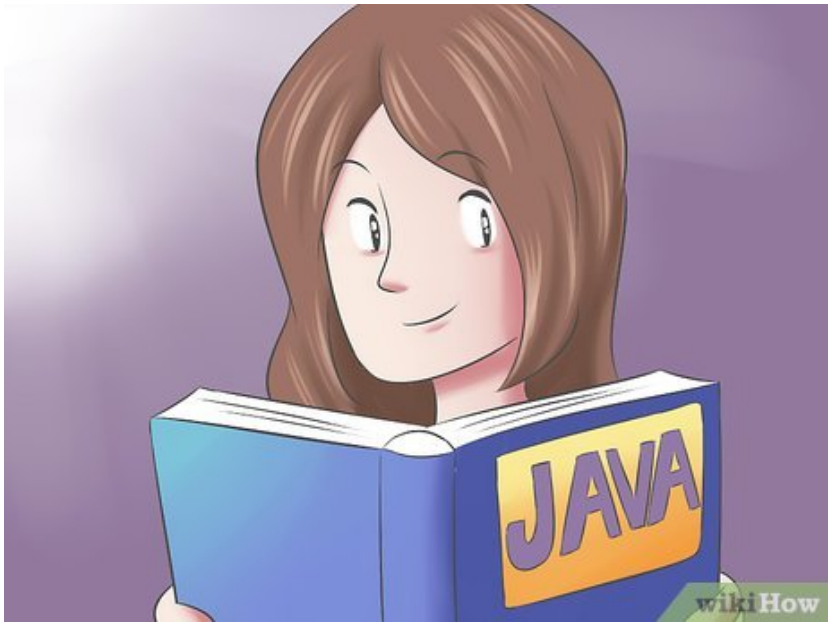
1. Some languages are interpreted languages, which means they don't need a compiler. Instead, they only need the language interpreter installed on the computer, and the programs can run instantly. Some examples of interpreted languages include Perl and Python.

4. 

Learn basic programming concepts. No matter which language you pick, you'll likely need to understand some basic common concepts. Knowing how to handle the syntax of the language will allow you to create much more powerful programs. Common concepts include:

1. Declaring variables - Variables are the way your data is temporarily stored in your program. This data can then be stored, modified, manipulated, and called upon later in the program.

2. Using conditional statements (if, else, when, etc.) - These are one of the basic functions of programs, and dictate how the logic works. Conditional statements revolve around "true" and "false" statements.
3. Using loops (for, goto, do, etc.) - Loops allow you to repeat processes over and over until a command is given to stop.
4. Using escape sequences - These commands perform functions such as create new lines, indents, quotes, and more.
5. Commenting on code - Comments are essential for remembering what your code does, helping other programmers understand your code, and for temporarily disabling parts of code.
6. Understand regular expressions.

5. 

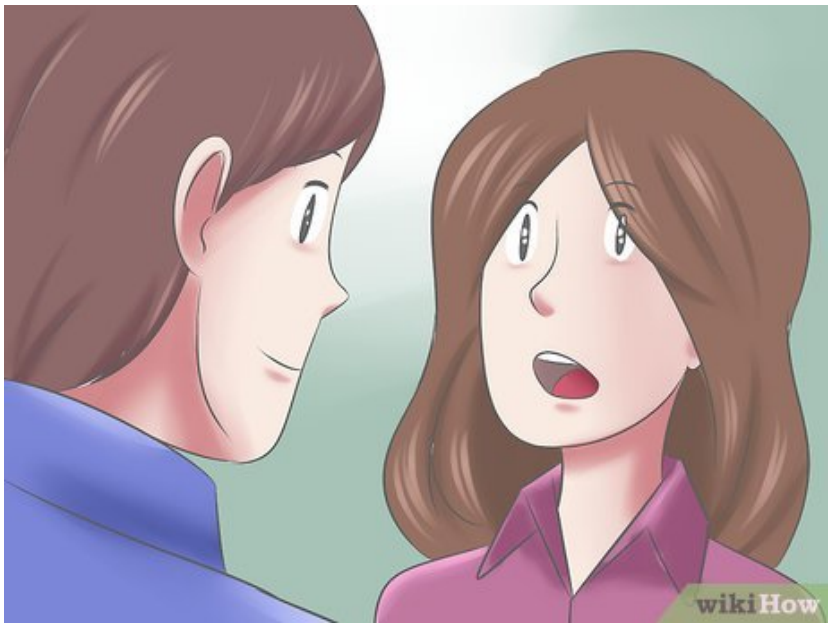
Find some books on the language of your choice. There are books for every language and for every level of expertise. You can find programming books at your local bookstore or any online retailer. A book can be an invaluable tool as you can keep it close at hand while you're working.

1. Beyond books, the internet is an endless treasure-trove of guides and tutorials. Search for guides on the language of your choice on sites such as Codecademy, Code.org, Bento, Udacity, Udemy, Khan Academy, W3Schools, and many more.

6. 

Take some classes. Anyone can teach themselves to make a program if they put their mind to it, but sometimes having a teacher and a classroom environment can be really beneficial. One-on-one time with an expert can greatly decrease the time it takes you to grasp programming fundamentals and concepts. Classes are also a good place to learn advanced math and logic that will be required for more complex programs.

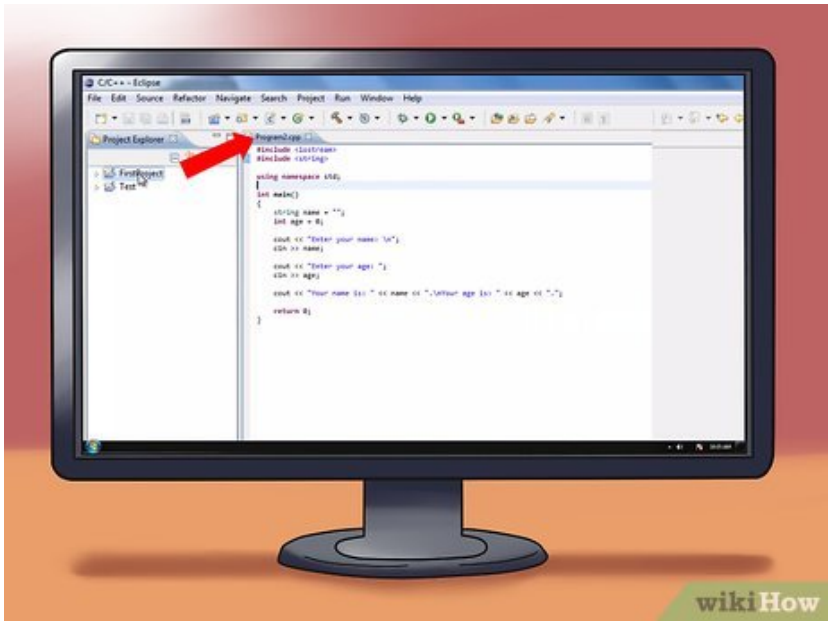
1. Classes cost money, so make sure that you are signing up for classes that will help you learn what you want to know.

7. 

Ask questions. The internet is a fantastic way to connect with other developers. If you find yourself stumped on one of your projects, ask for help on sites such as StackOverflow. Make sure that you ask in an intelligent manner and can prove that you have already tried several possible solutions.

Part 3 of 6:

Building Your Prototype

1. 

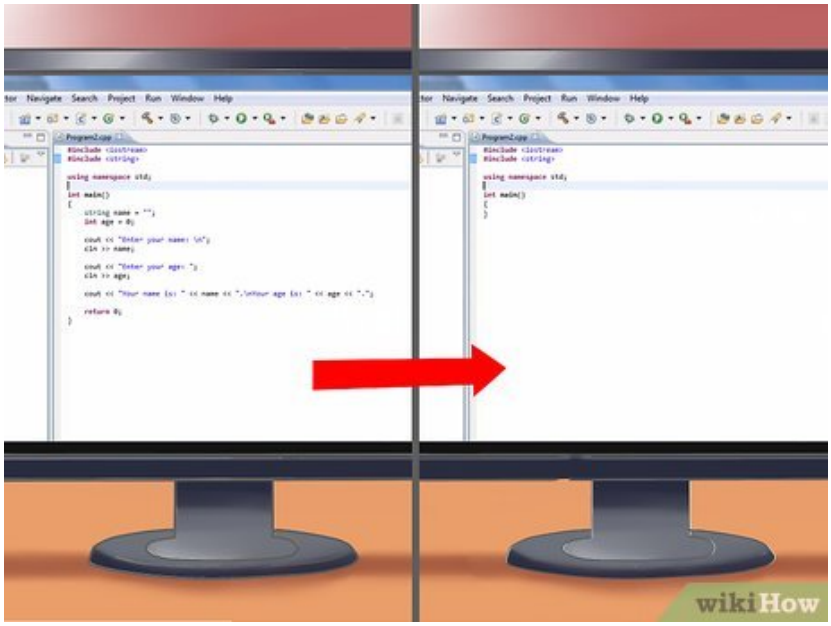
Start writing a basic program with your core functionality. This will be the prototype that shows off the functionality that you're aiming to achieve. A prototype is a quick program, and should be iterated on until you find a design that works. For example, if you are creating a calendar program, your prototype would be a basic calendar (with correct dates!) and a way to add events to it.

1. As you create your prototype, use a top-down approach. Leave out as much detail as you possibly can at first. Then, slowly add finer and finer details. This will speed up the prototyping process and will also keep your code from getting too complex and unmanageable. If your code gets too hard to follow, you could end up having to start all over from the beginning.
2. Your prototype will change often during the development cycle as you come up with new ways to tackle problems or think of an idea later that you want to incorporate.
3. If you're making a game, your prototype should be fun! If the prototype isn't fun, then chances are the full game won't be fun either.
4. If your desired mechanics just aren't working in the prototype, then it may be time to go back to the drawing board.

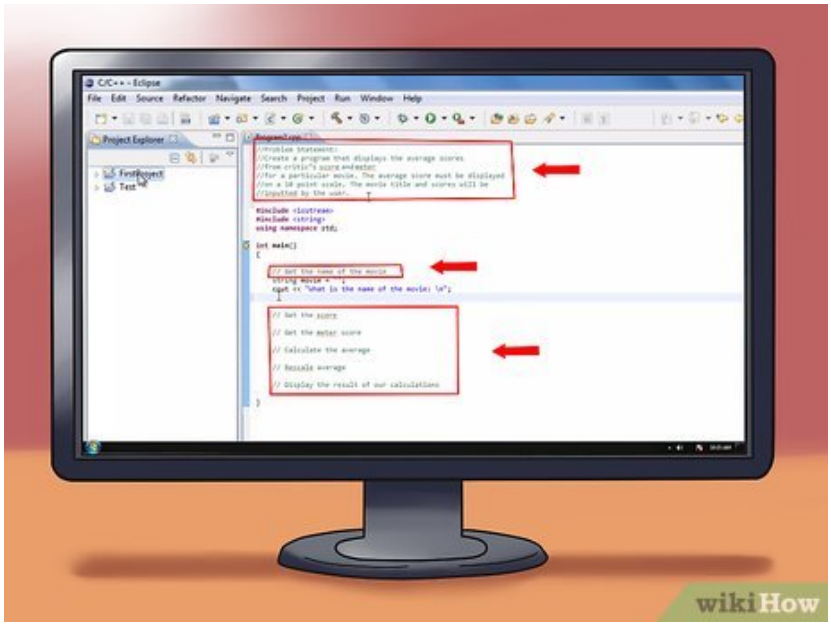
2. 

Assemble a team. If you are developing your program on your own, you can use a prototype to help build a team. A team will help you track down bugs faster, iterate features, and design the visual aspects of the program.

1. A team is definitely not necessary for small projects, but will cut down development time significantly.
2. Running a team is a complex and difficult process, and requires good management skills along with a good structure for the team. See this guide for more details on leading a group.

3. 

Start over from scratch if necessary. Once you're familiar with your language, you may be able to get prototypes up and running in just a few days. Because of their quick nature, don't be afraid to scrap your idea and start over from a different angle if you're not happy with how it's turning out. It's much easier to make major changes at this stage than it is later on when the features start falling into place.

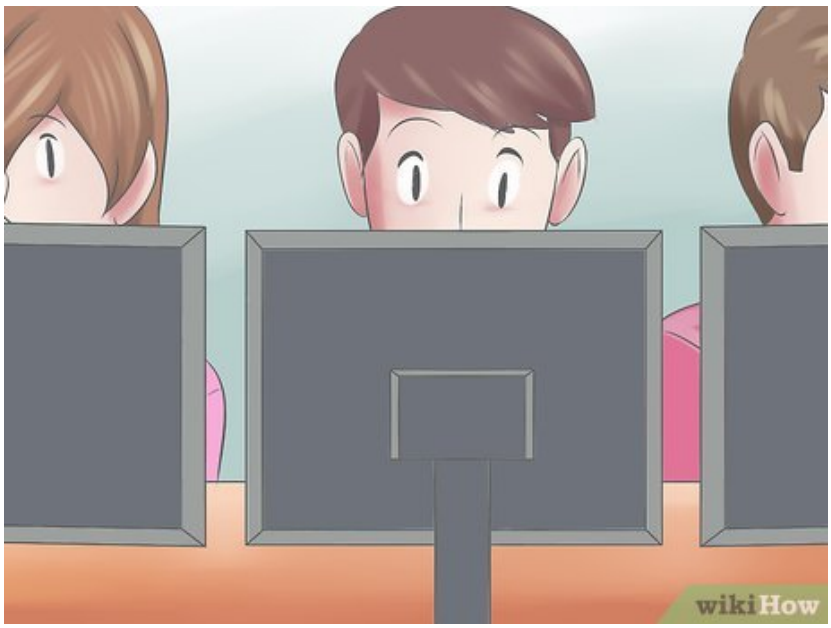
4. 

Comment on everything. Use the comment syntax in your programming language to leave notes on all but the most basic lines of code. This will help you remember what you were doing if you have to put the project down for awhile, and will help other developers understand your code. This is especially essential if you are working as part of a programming team.

1. You can use comments to temporarily disable parts of your code during testing. Simply enclose the code you want to disable in comment syntax and it won't be compiled. You can then delete the comment syntax and the code will be restored.

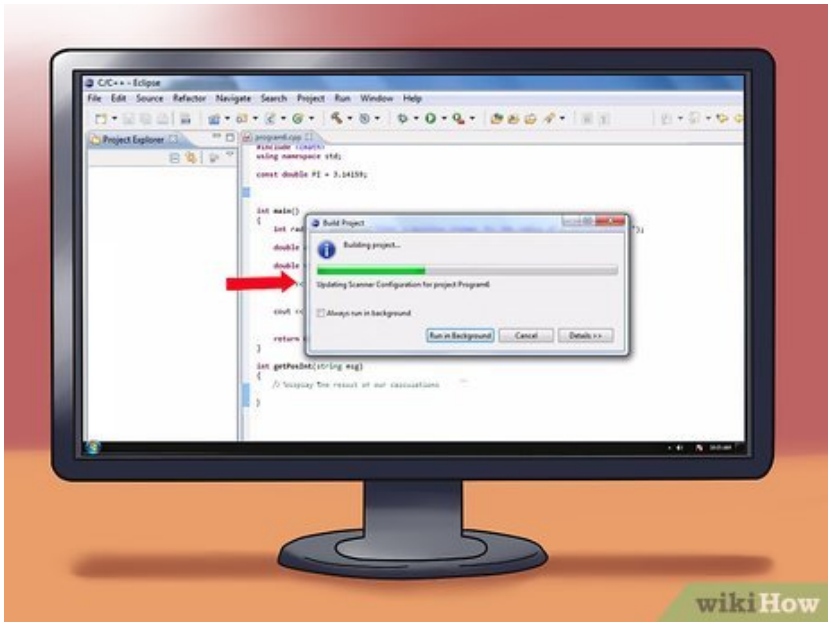
Part 4 of 6:

Alpha Testing

1. 

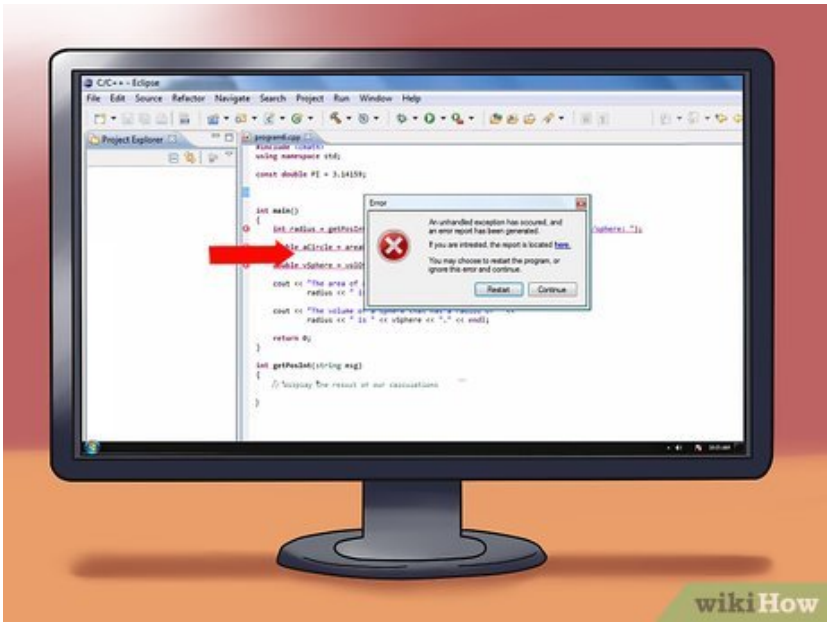
Gather a testing team. In the alpha stage, the testing team can and should be small. A small group will help you get focused feedback and gives you the ability to interface with testers one on one. Every time you make updates to the prototype, new builds are sent out to the alpha testers. The testers then try all of the included features and also try to break the program, documenting their results.

1. If you are developing a commercial product, you will want to make sure that all of your testers sign a Non-Disclosure Agreement (NDA). This will prevent them from telling others about your program, and prevent leaks to press and other users.
2. Take some time to come up with a solid testing plan. Make sure that your testers have a way to easily report bugs in the program, as well as easily access new versions of the alpha. GitHub and other code repositories are a great way to easily manage this aspect.

2. 

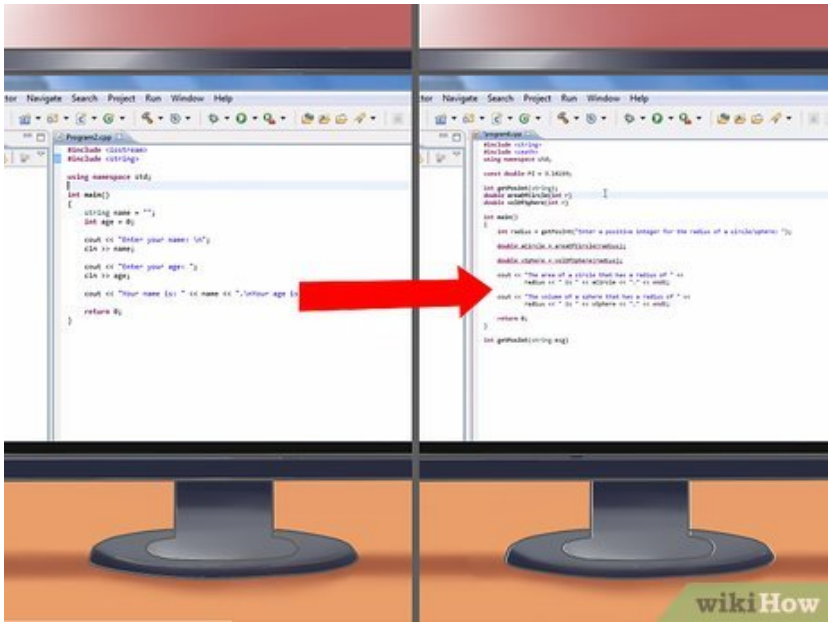
Test your prototype over and over. Bugs are the bane of every developer. Errors in code and unexpected usage can cause all kinds of problems in a finished product. As you continue to work on your prototype, test it as much as possible. Do everything you can to break it, and then try to keep it from breaking in the future.

1. Try inputting odd dates if your program deals with dates. Really old dates or far future dates may cause odd reactions with the program.
2. Input the wrong kind of variables. For example, if you have a form that asks for the user's age, enter in a word instead and see what happens to the program.
3. If your program has a graphical interface, click on everything. What happens when you go back to a previous screen, or click buttons in the wrong order?

3. 

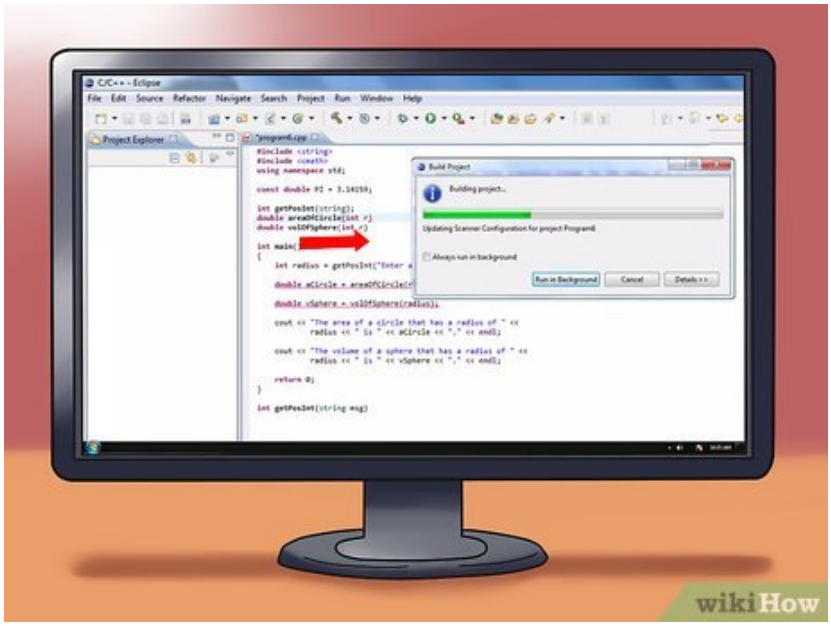
Address bugs in order of priority. When revising the program in the alpha, you will be spending a lot of time fixing features that do not work correctly. When organizing your bug reports from your alpha testers, they will need to be sorted based on two metrics: **Severity** and **Priority**.

1. The Severity of a bug is a measure of how much damage the bug causes. Bugs that crash the program, corrupt data, keep the program from running are referred to as Blockers. Features that don't work or return incorrect results are labeled Critical, while difficult to use or bad-looking features are labeled Major. There are also Normal, Minor, and Trivial bugs that affect smaller sections or less-crucial features.
2. The Priority of a bug determines what order you tackle them when attempting to fix bugs. Fixing bugs in software is a time-consuming process, and takes away from the time you have to add features and polish. As such, you have to take the priority of a bug into account to make sure that you meet deadlines. All Blocker and Critical bugs take the highest priority, sometimes referred to as P1. P2 bugs are usually Major bugs that are scheduled to be fixed, but won't hold a product back from being shipped. P3 and P4 bugs are usually not scheduled fixes, and fall into the "nice to have" category.

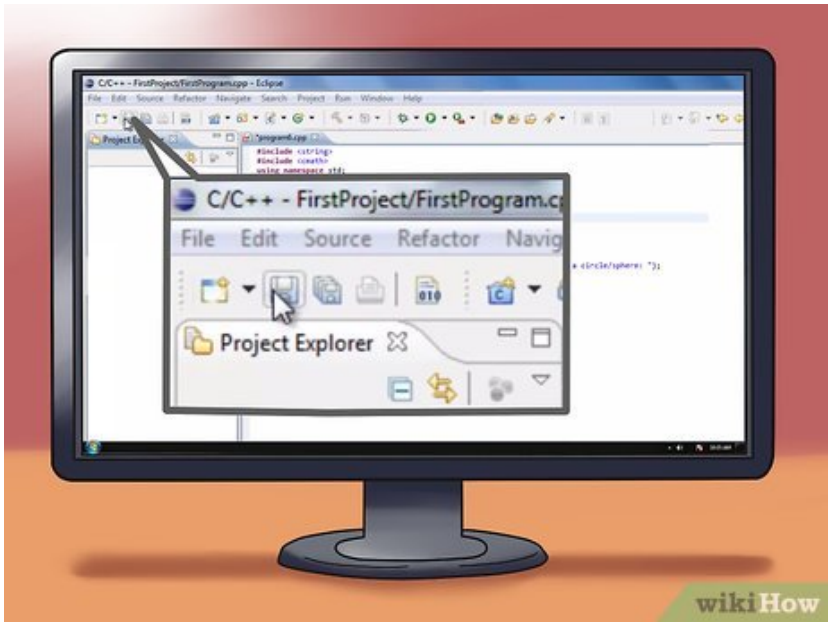
4. 

Add more features. During the alpha phase, you will be adding more features to your program to bring it closer to the program outlined in your design document. The alpha stage is where the prototype evolves into the basic for of the full program. By the end of the alpha stage, your program should have all of its features implemented.

1. Don't stray too far from your original design document. A common problem in software development is "feature-creep", where new ideas keep getting added, causing the original focus to be lost and spreading development time between too many different features. You want your program to be the best at what it does, not a jack of all trades.[2]

5. 

Test each feature as you add it. As you add features to your program during the alpha phase, send out the new build to your testers. The regularity of new builds will be entirely dependent on your team's size and how much progress you're making on the features.

6. 

Lock your features when the alpha is finished. Once you've implemented all of the features and functionality in your program, you can move out of the alpha phase. At this point, no further features should be added, and the included features should essentially work. Now you can move onto wider testing and polish, known as the beta phase.

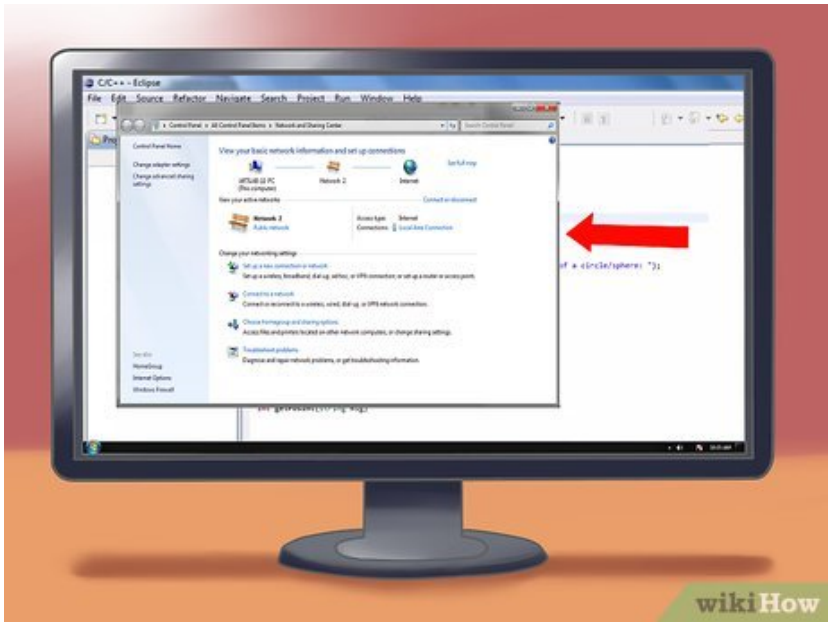
Part 5 of 6:

Beta Testing

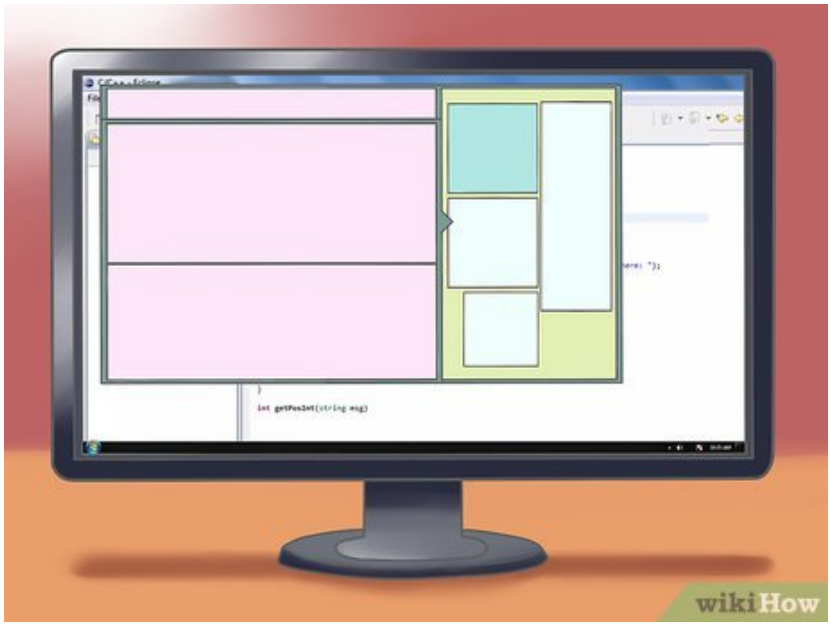
1. 

Increase your testing group size. In the beta phase, the program is made available to a much larger group of testers. Some developers make the beta phase public, which is referred to as an open beta. This allows anyone to sign up and participate in testing the product.

1. Depending on the needs of your product, you may or may not want to do an open beta.

2. 

Test connectivity. As programs become more and more interconnected, there's a good chance that your program will rely on connections to other products or connections to servers. Beta testing allows you to ensure that these connections work under a larger load, which will ensure that your program is usable by the public when it releases.

3. 

Polish your software. In the beta phase, no more features are being added, so focus can be turned to improving the program's aesthetics and usability. In this phase, UI design becomes a priority, ensuring that the users will have no difficulty navigating the program and taking advantage of the features.

1. UI design and functionality can be very difficult and complex. People make whole careers out of designing UIs. Just make sure that your personal project is easy to use and easy on the eyes. A professional UI may not be possible without a budget and a team.

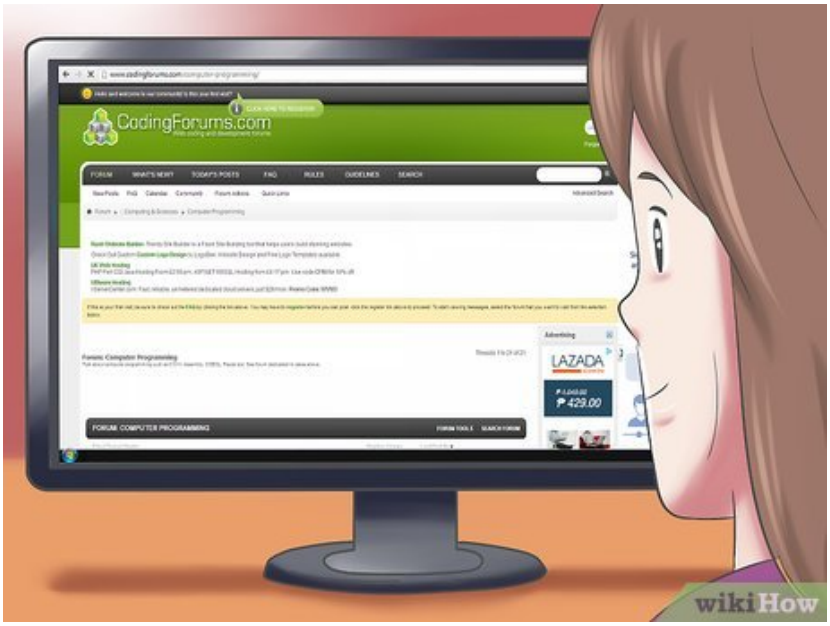
2. If you have the budget, there are lots of freelance graphics designers who could potentially design a UI on contract for you. If you have a solid project that you're hoping will become the next big thing, find a good UI designer and make them part of your team.

4. 

Continue bug hunting. Throughout the beta phase, you should still be cataloging and prioritizing bug reports from your user base. Since more testers will have access to the product, chances are new bugs will be discovered. Eliminate bugs based on their priority, keeping your final deadlines in mind.[3]

Part 6 of 6:

Releasing the Program

1. 

Market your program. If you want to get users, you'll want to make sure that they know your program exists. Just like any product, you'll need to do a bit of advertising in order to make people aware. The extent and depth of your marketing campaign will be dictated by your program's function as well as your available budget. Some easy ways to raise awareness of you program include:

1. Posting about your program on related message boards. Make sure that you follow the posting rules of whichever forum you choose so that your posts don't get marked as spam.
2. Send out press releases to tech sites. Find some tech blogs and sites that fit your program's genre. Send the editors a press release detailing your program and what it does. Include a few screenshots.
3. Make some YouTube videos. If your program is designed to complete a specific task, make some YouTube videos showing your program in action. Structure them as "How-To" videos.
4. Create social media pages. You can create free Facebook and Google+ pages for your program, and can use Twitter for both company and program-specific news.

2. 

Host your program on your website. For small programs, you can most likely host the file on your own website. You may want to include a payment system if you are going to be charging for your software. If your program becomes very popular, you may need to host the file on a server that can handle more downloads.

3. 

Set up a support service. Once your program is released in the wild, you will invariably have users with technical problems or who don't understand how the program works. Your website should have thorough documentation available, as well as some sort of support service. This can include a technical support forum, a support email, live help, or any combination of those. What you can provide will be dependent on your available budget.

4. 

Keep your product up to date. Almost all programs these days are patched and updated long after their initial release. These patches may fix critical or non-critical bugs, update security protocols, improve stability, or even add functionality or redo the aesthetics. Keeping your program updated will help keep in competitive.

Sample Programs

Picture 30 of How to Create a Program

Sample C++ Program

Picture 31 of How to Create a Program

Sample MATLAB Programs