Tutorial – Variables: Zorp part 1

Introduction:

Over the next several tutorials we will put the topics we learn into practice by making a text-based adventure game.

These games were quite common in the early days of personal computing, and one of the most successful of these was the game *Zork* (first published in 1977). While our tutorial game won't come close to the detail and complexity of a game like *Zork*, it will still provide us with a window into what goes into making a (modern) computer game.

Our reason for focusing on a text adventure game (as opposed to something more visual and exciting) is to allow you to focus on gaining competence in the C++ programming language without the distractions of the complex engine and asset pipelines found in modern game development workflows.

By the end of this subject (and this tutorial series), you should have gained enough familiarity with C++ to attempt these more complex, 2D and 3D computer games.

For now, let's begin our journey with a look at one of the most fundamental tasks in programming: retrieving user input.

In this tutorial, we'll look at how to prompt the player for information, correctly read and interpret their input, store the input in variables to use during processing, and output the result of this processing back to the player.

Creating a New Project:

The first step is creating a new Visual Studio C++ project. If you have not yet installed Visual Studio, you can download the Community Edition from this site: https://www.visualstudio.com/vs/community/.

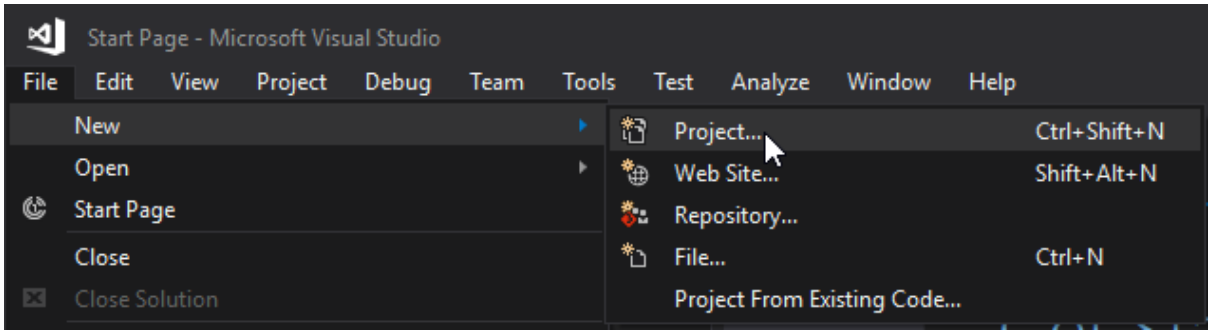
Your teacher will inform you if you need to install the 2015 or 2017 version. There are differences between the two, and while projects are for the most part compatible, it is likely you will run into trouble if you install the wrong version. If you are not sure which version to use, please ask.

For our purposes, there is no difference between the Community Edition and the Professional or Enterprise versions, but if you are curious you can find a comparison here: https://www.visualstudio.com/vs/compare/.

If you have an AIE email account, or a student validated Microsoft account, you can download the other versions (as well as other Microsoft products) for free from this site: https://imagine.microsoft.com/.

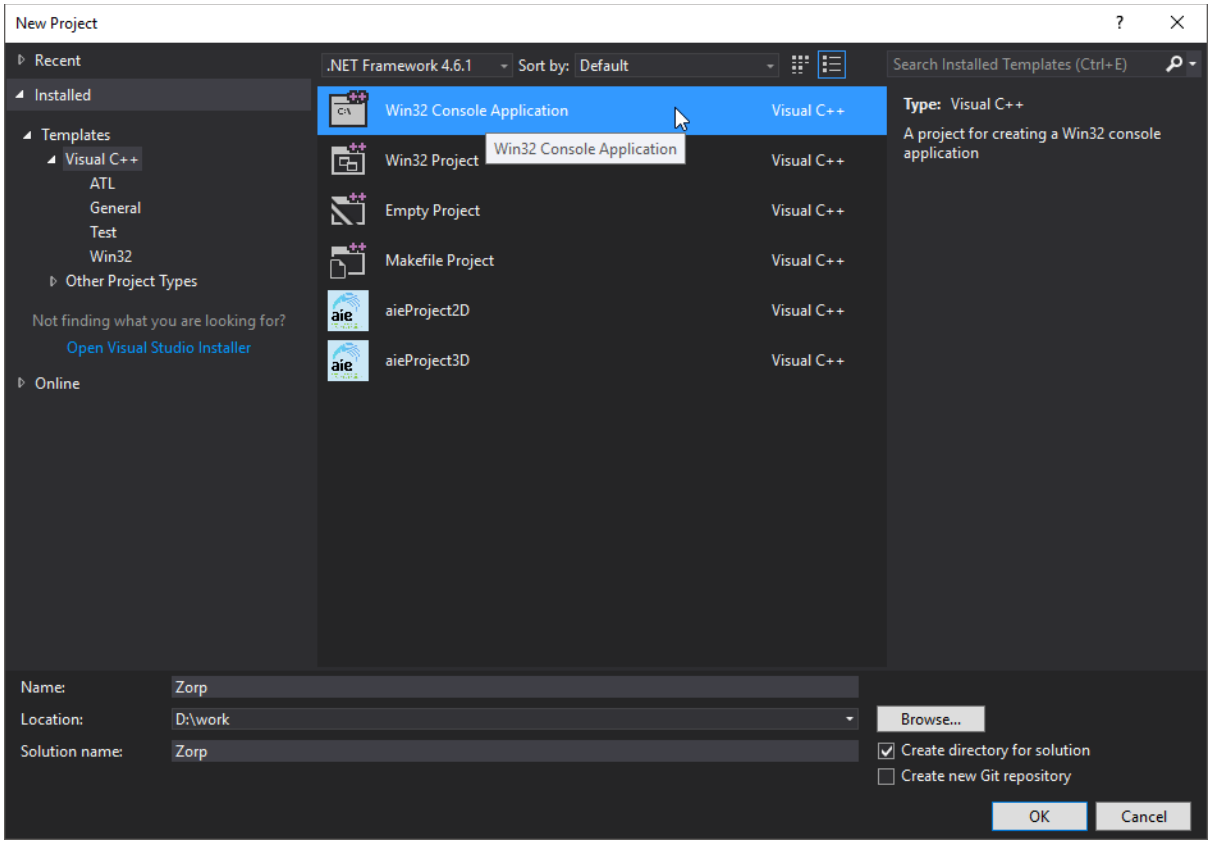
© AIE 2017

Once you have Visual Studio installed and open, select the *File -> New -> Project…* option.



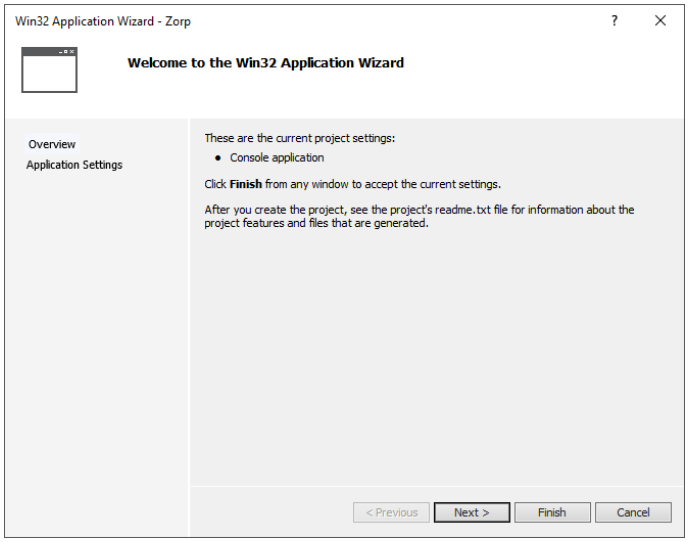
Select *Win32 Console Application*. (Don't worry if you're using an x64 machine, it will still work.)

Fill in the *Name* and *Location* fields, and press *OK*.



© AIE 2017

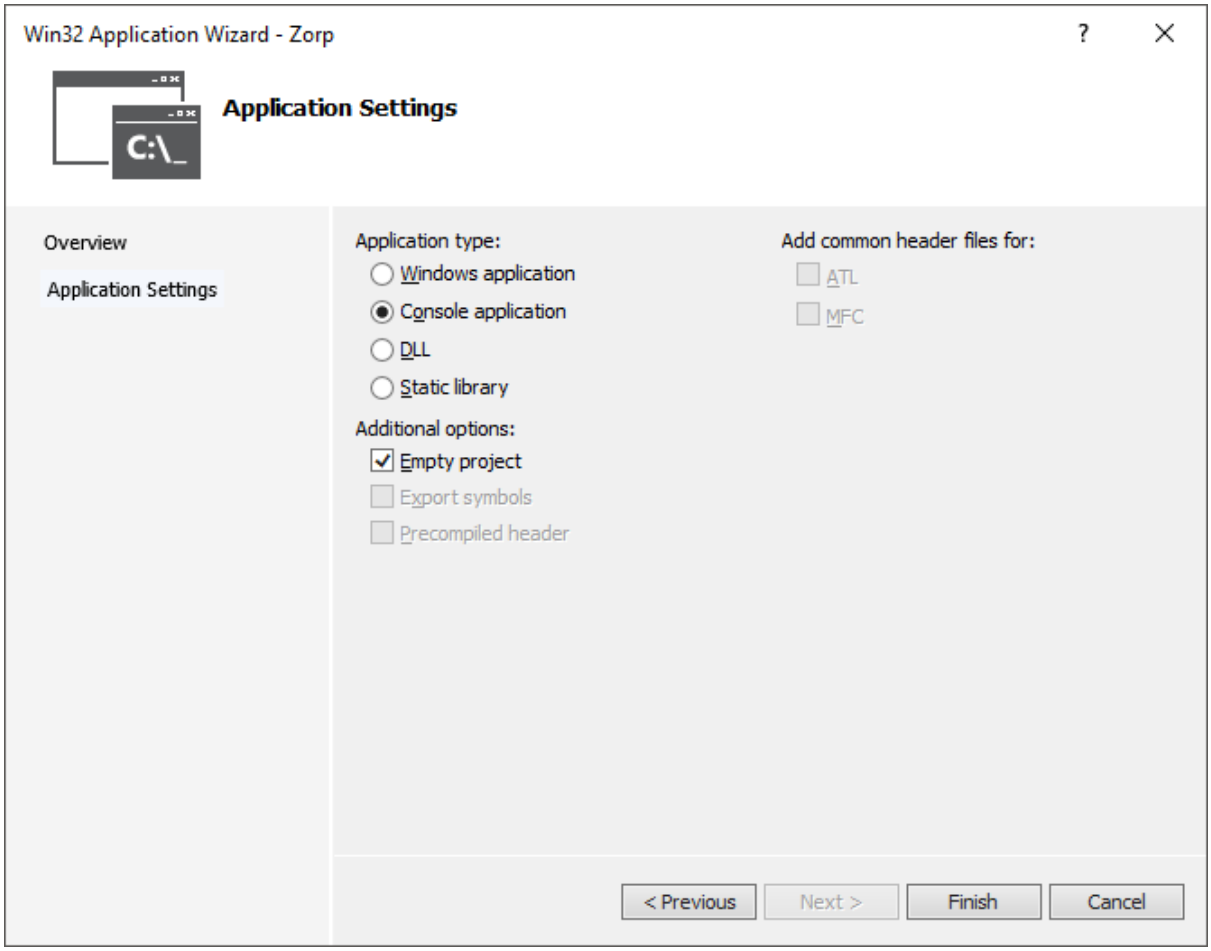
The next screen just confirms you wish to make a Win32 console application. Press *Next*.



In the final screen, ensure *Console Application* is selected. (*Windows Application* is for making a form-based application, and will not suit our needs.)

In the *Additional Options*, deselect *Precompiled header* and select *Empty project*.

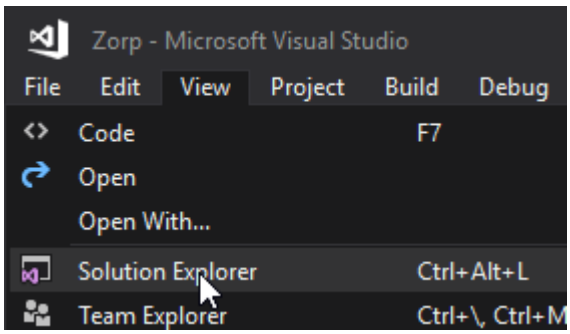
Once you have set all the options, press *Finish*.



© AIE 2017

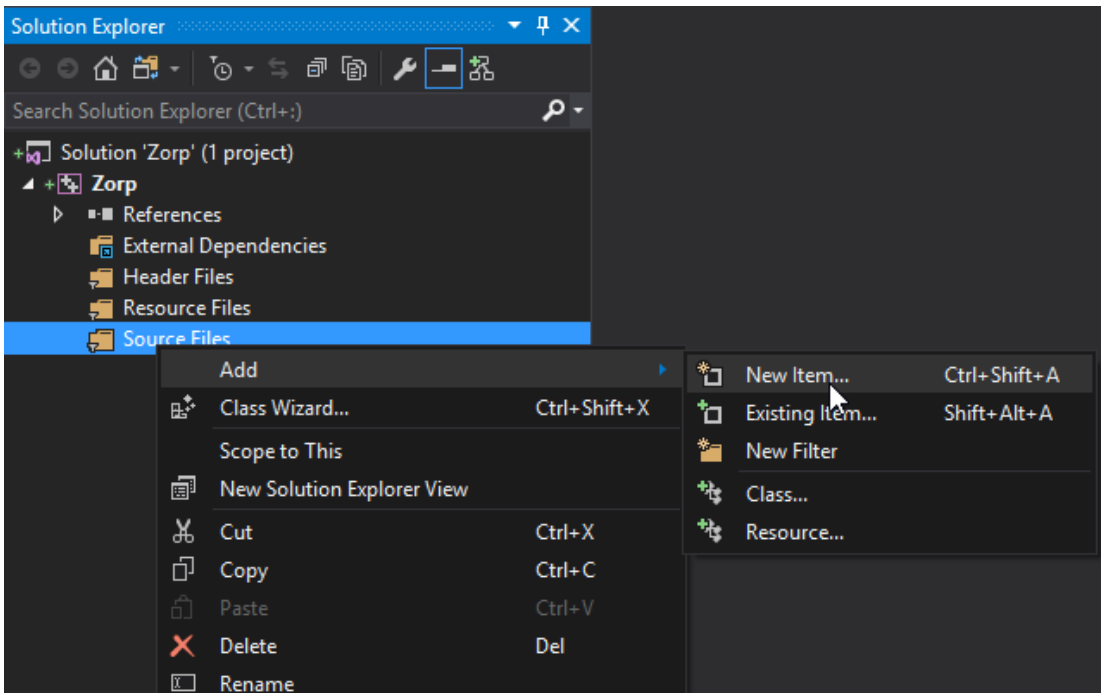
Your project will now be created and opened in Visual Studio.

The main window we use to navigate the files in the project is the *Solution Explorer*. If you do not see this window, select the *View -> Solution Explorer* option.



The *Solution Explorer* is opened on the right-side of the screen by default (after a fresh install). You can drag the window around and dock it anywhere you like.

Locate the *Source Files* folder and right-click. From the context menu, select *Add -> New Item…*

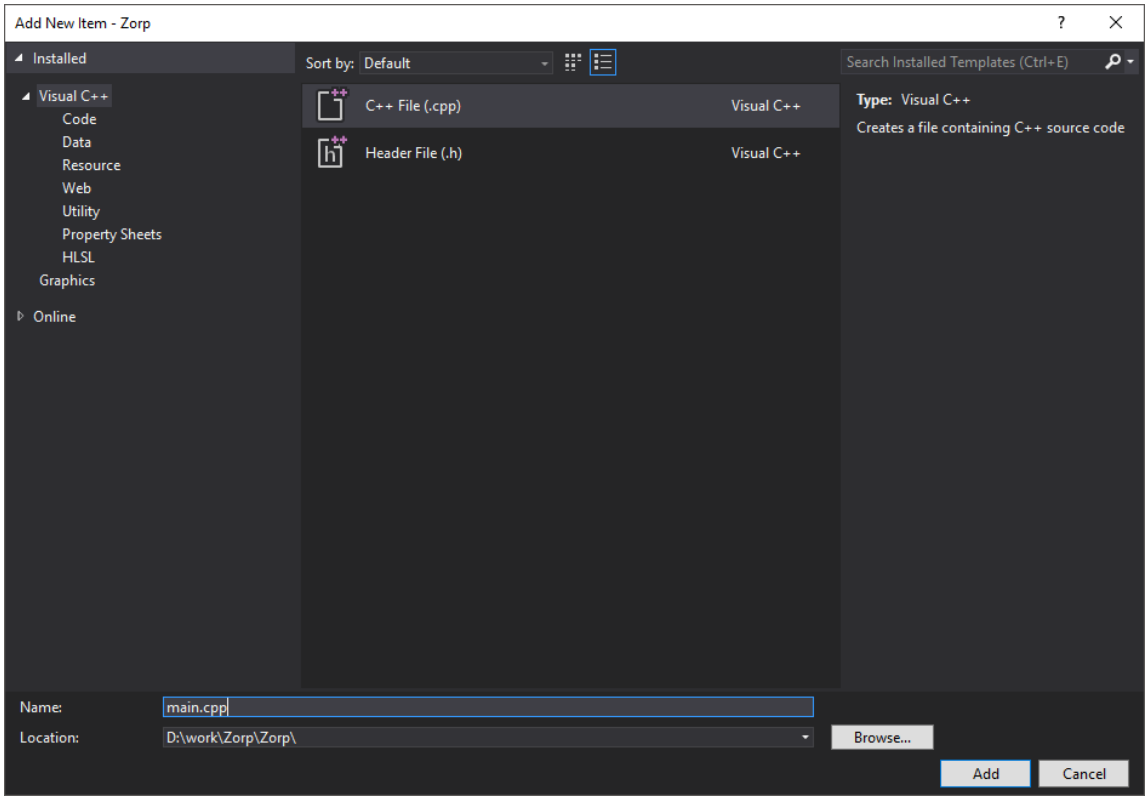


This will allow us to create our first source code file. (Every project needs at least one source code file).

From the dialog window that opens, select *C++ File (.cpp)*.

In the *Name* field, enter *main.cpp* as the name of the file to create, and press the *Add* button.

© AIE 2017



The blank *main.cpp* file will then be created and opened in the editor.

We're now ready to begin programming our game.

Hello World:

The core of our game will be writing to, and reading from the console window (also called the command line). To do this we need to access the *std::cout* and *std::cin* streams (for output and input respectively). The definitions of these stream objects is in the *iostream* header file. To use them, we need to include the header in out project.

Add the following line to the top of the *main.cpp* file:

```
#include <iostream>
```

We now need to make the *main()* function.

Every program you write in C++ begins with the *main()* function. (In fact, all C++ programs *must* have one – otherwise the computer doesn't know where the start of your program is).

If you're familiar with C++ you may know that there are a few different ways we can define this function, depending on the functionality we want. If we wanted out program to use run-time

© AIE 2017

arguments, we'd add function arguments. If we wanted our program to return an error code, we'd specify a return type. We don't need either of these options at the moment, so we'll define our *main()* function as follows:

```
#include <iostream>

void main()
{

}
```

If you compile and run the project now (by pressing *F5* or by selecting *Debug -> Start Debugging*) you'd notice that our program doesn't do anything (apart from opening and immediately closing a console window).

Let's change that by adding some guts to the program:

```
#include <iostream>

void main()
{
	std::cout << "hello world!";
	return;
}
```

Now, if you're very quick, you might see the message '*Hello world!*' flash on the screen before the program ends and the console window closes.

Waiting for Input:

So, what happened?

Our program immediately closed before we could see the output.

If we were running the executable from a console window as a stand-alone program, the window would remain open after the program ends, so this wouldn't be a problem.

But when we run our programs from the Visual Studio editor, w once they end Visual Studio will clean everything up for us and close any windows it opened.

We'd like to pause our program just before it exits, and wait for the player to press *Enter* before the program shuts down.

Many people will add the following line of code immediately before the *return;* statement to pause the program and wait for player input:

© AIE 2017

```
#include <iostream>

void main()
{
	std::cout << "hello world!";

	system("pause");
	return;
}
```

What happens on this line is we tell Windows to execute one of its system command (namely, the pause command). The command will execute and wait for input before returning control back to our program, at which point we exit and shutdown the program. It is a simple and effective way of adding a pause to your program.

While this code isn't inherently bad, it does suffer from one fatal flaw: it will only work on Windows machines.

If you take this same program and compile it in Linux (or Mac), that one line of code will prevent the whole program from compiling. For some developers, this is a deal-breaker.

A platform independent way of adding a pause to your game is as follows:

```
#include <iostream>

void main()
{
	std::cout << "hello world!";

	std::cout << std::endl << "Press 'Enter' to exit the program." << std::endl;
	std::cin.get();
	return;
}
```

Here we've using the *cin* object (which will read input from the command line) to pause the execution of the program while it waits for the player to enter input. We don't actually do anything with the input (in fact, we don't care what the input is), we just wait for any input before exiting.

We've also added a line to tell the player to press *Enter*.

There are still problems with using *cin.get()* to wait for input, but we'll address those a little later. For now, run the program and test its functionality.

Storing Input in Variables:

Now we're going to add some variables to our game, and fill them with the values that the player enters via the console.

© AIE 2017

Update your *main.cpp* file as follows:

```
#include <iostream>

void main()
{
	std::cout << "hello world!";

	int height = 0;

	std::cout << "Welcome to ZORP!" << std::endl;
	std::cout << "ZORP is a game of adventure, danger, and low cunning."
		<< std::endl;
	std::cout << "It is definitely not related to any other text-based adventure
		game." << std::endl << std::endl;

	std::cout << "First, some questions..." << std::endl;
	std::cout << "How tall are you, in centimeters? " << std::endl;

	std::cin >> height;

	std::cout << "You entered " << height << std::endl;

	std::cout << std::endl << "Press 'Enter' to exit the program." << std::endl;
	std::cin.get();
	return;
}
```

In our update we first display a bit of introductory text (no text-based adventure game worth its name would be without it), before asking the player to enter their height.

This input is read and stored in the *height* variable. Finally, we output the height value back to the console.

If you run your game now, you'll notice the program exits immediately after you input the height value. Why?

What actually happens is that the input value (I hope you entered only numbers) is stored in the *height* variable. But because this variable is an integer, the *Enter* keypress (i.e., the new-line character) remains in the input buffer (because it's the first non-numeric character).

Then, we call *cin.get()* again just before the *return* statement. Because there is an *Enter* keypress remaining in the buffer, the program thinks the player just pressed *Enter* and so immediately exits.

What can we do about this? We need to clear the buffer after each value is read from the command line.

Update your program as follows:

© AIE 2017

```
#include <iostream>

void main()
{
	std::cout << "hello world!";

	int height = 0;

	std::cout << "Welcome to ZORP!" << std::endl;
	std::cout << "ZORP is a game of adventure, danger, and low cunning."
		<< std::endl;
	std::cout << "It is definitely not related to any other text-based adventure
		game." << std::endl << std::endl;

	std::cout << "First, some questions..." << std::endl;
	std::cout << "How tall are you, in centimeters? " << std::endl;

	std::cin >> height;
	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());

	std::cout << "You entered " << height << std::endl;

	std::cout << std::endl << "Press 'Enter' to exit the program." << std::endl;
	std::cin.get();
	return;
}
```

We call *cin.clear()* to clear any error flags that may have been set on the input buffer (for example, error flags will be set when we try to read characters and store them in an integer variable).

Then *cin.ignore()* is called to clear all the characters from the input buffer. *cin.ignore()* needs an argument that tells it how many characters to ignore, so we use *cin.rdbuf()->in_avail()* to read how many characters remain in the buffer and pass this value to *cin.ignore()*.

These two lines solve the problem I alluded to before with using *cin.get()* for pausing the program. As long as there are no characters remaining to be read in the input buffer, using *cin.get()* to pause the program and wait for input won't be a problem (and the code is more portable).

Run your program again. Everything should be fine if you enter numbers when prompted for your height. Try entering random characters and see what happens.

We can check if the user entered valid input by checking the error flags on the *cin* object. Update the code as follows:

© AIE 2017

```
#include <iostream>

void main()
{
	int height = 0;

	std::cout << "Welcome to ZORP!" << std::endl;
	std::cout << "ZORP is a game of adventure, danger, and low cunning."
		<< std::endl;
	std::cout << "It is definitely not related to any other text-based adventure
		game." << std::endl << std::endl;

	std::cout << "First, some questions..." << std::endl;
	std::cout << "How tall are you, in centimeters? " << std::endl;

	std::cin >> height;
	if (std::cin.fail()) {
		std::cout << "You have failed the first challenge and are eaten
			by a grue." << std::endl;
	}
	else {
		std::cout << "You entered " << height << std::endl;
	}
	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());

	std::cout << std::endl << "Press 'Enter' to exit the program." << std::endl;
	std::cin.get();
	return;
}
```

We've updated our code so now we can tell if the player tried to enter something that wasn't a number when prompted for their height.

You may not have formally covered *If* statements yet. If not, consider this a preview for what is to come.

The *If* statement is simply checking the fail bit of the *cin* object. The *fail()* function will be true if anything failed while attempting to read input from the command line and store it in the *height* variable. The most common cause of error will be entering non-numeric text, but there are a few other things that could cause *cin* to fail to read (but these mostly relate to working with files instead of the command line).

What about if we wanted to read a character and store it in a variable? I'm glad you asked. Update your program as follows:

© AIE 2017

```
#include <iostream>

void main()
{
	int height = 0;
	char firstLetterOfName = 0;

	std::cout << "Welcome to ZORP!" << std::endl;
	std::cout << "ZORP is a game of adventure, danger, and low cunning."
			<< std::endl;
	std::cout << "It is definitely not related to any other text-based adventure
			game." << std::endl << std::endl;

	std::cout << "First, some questions..." << std::endl;
	std::cout << "How tall are you, in centimeters? " << std::endl;

	std::cin >> height;
	if (std::cin.fail()) {
		std::cout << "You have failed the first challenge and are eaten by a
				grue." << std::endl;
	}
	else {
		std::cout << "You entered " << height << std::endl;
	}
	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());

	std::cout << "What is the first letter of your name? " << std::endl;

	std::cin >> firstLetterOfName;

	if (std::cin.fail() || !isalpha(firstLetterOfName)) {
		std::cout << "You have failed the second challenge and are eaten by a
				grue." << std::endl;
	}
	else {
		std::cout << "You entered " << firstLetterOfName << std::endl;
	}
	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());


	std::cout << std::endl << "Press 'Enter' to exit the program." << std::endl;
	std::cin.get();
	return;
}
```

The code is much the same. You'll see that first we try to store the input in the variable *firstLetterOfName*.

We then call the *fail()* function to see if that worked. You'll also notice the code `!isalpha(firstLetterOfName)`. This function checks whether or not the input variable is an alpha character (a..z). If the character was not *alpha*, then we have an error.

© AIE 2017

The reason we need this additional check is because numbers are considered characters (or rather, they have character equivalents). So by entering a number the program will interpret this as a character representing a number (i.e, '1' instead of 1). The difference is subtle, but very important.

Using the Variables:

Having stored values in our program isn't very helpful unless we actually use those values to do some sort of processing. So, let's add a bit of calculation to our program.

Update the *main.cpp* file as follows:

© AIE 2017

```
#include <iostream>

void main() {
	int height = 0;
	char firstLetterOfName = 0;
	int avatarHP = 0;

	std::cout << "Welcome to ZORP!" << std::endl;
	std::cout << "ZORP is a game of adventure, danger, and low cunning."
			<< std::endl;
	std::cout << "It is definitely not related to any other text-based adventure
			game." << std::endl << std::endl;

	std::cout << "First, some questions..." << std::endl;
	std::cout << "How tall are you, in centimeters? " << std::endl;
	std::cin >> height;
	if (std::cin.fail()) {
		std::cout << "You have failed the first challenge and are eaten by a
				grue." << std::endl;
	}
	else {
		std::cout << "You entered " << height << std::endl;
	}
	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());

	std::cout << "What is the first letter of your name? " << std::endl;
	std::cin >> firstLetterOfName;
	if (std::cin.fail() || !isalpha(firstLetterOfName)) {
		std::cout << "You have failed the second challenge and are eaten by a
				grue." << std::endl;
	}
	else {
		std::cout << "You entered " << firstLetterOfName << std::endl;
	}
	std::cin.clear();
	std::cin.ignore(std::cin.rdbuf()->in_avail());

	if (firstLetterOfName != 0) {
		avatarHP = (float)height / (firstLetterOfName * 0.02f);
	}
	else {
		avatarHP = 0;
	}

	std::cout << std::endl << "Using a complex deterministic algorithm, it has
			been calculated that you have " << avatarHP << " hit
			point(s)." << std::endl;

	std::cout << std::endl << "Press 'Enter' to exit the program." << std::endl;
	std::cin.get();
	return;
}
```

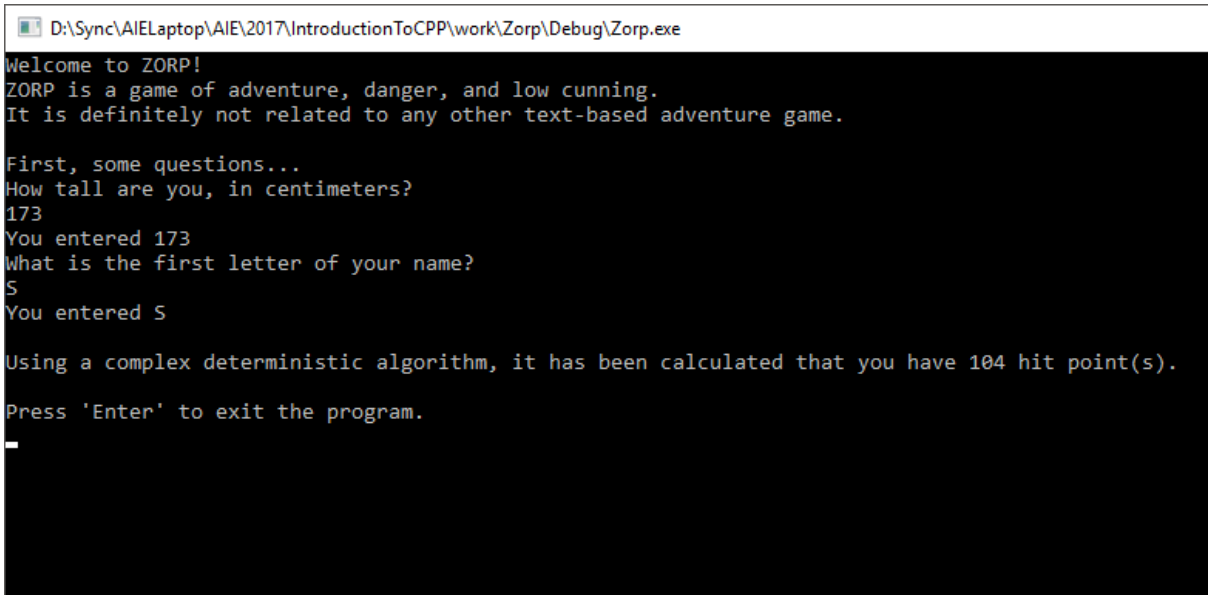
In this final piece of code, we do a simple calculation using the values previously entered to come up with a value for the player's HP.

Before performing this calculation we first check if the value of *firstLetterOfName* is 0. If this value is ever 0 then the following divide operation would crash the program (because the result of dividing by 0 cannot be calculated – go ahead, try it!).

© AIE 2017

Also notice that we're using a *char* variable as if it were a normal integer. This is absolutely fine and something you would have learnt about during the lecture.

That's it. That's our game so far. Run your program and see verify that whatever input you enter will be handled correctly.



Before you pack away your code, take a moment to look at the following pages that further explain some of the functions we used in this tutorial:

- std:cin >>
  http://www.cplusplus.com/reference/istream/istream/operator%3E%3E/
- std::cin.get()
  http://www.cplusplus.com/reference/istream/istream/get/
- std::cin.clear()
  http://www.cplusplus.com/reference/ios/ios/clear/
- std::cin.ignore()
  http://www.cplusplus.com/reference/istream/istream/ignore/
- std::cin.fail()
  http://www.cplusplus.com/reference/ios/ios/fail/
- std::cout <<
  http://www.cplusplus.com/reference/ostream/ostream/operator%3C%3C/
- std::endl
  http://www.cplusplus.com/reference/ostream/endl/

© AIE 2017