



Introduction to C++ Programming

Thomas McElroy
May 5th, 2021

Before We Begin....



To follow along and have all the example codes that I have created, please grab the git repository and startup the docker image with the folder attached.

- `$git clone https://github.com/thomasphys/sapcppexamples.git`
- `$sudo docker run -it -v /home/tmcelroy/sapcppexamples/:/home/physuser/cppexamples:rw
soudk/eieioo2021`

** remember to change the path to your sapcppexamples directory.

Introduction:



- Built as an extension to C
- Added classes which are a backbone of C++
- Robust and is used for the backend of most programs.
- Unlike languages like python, c++ has statically typed variables.
- For the languages commonly used in physics, it is known for its use of pointers.

A few basics (bare with me):



Basic variable types:

- float - real number with precision of 4 bytes
 - float variable = 3.14;
- double - real number with precision of 8 bytes
 - double variable = 3.14;
- int - integer number of precision of 4 bytes
 - int i = -5;
- unsigned int - positive (and zero) integer.
 - unsigned int = 5;
- char - character variable ("a", "b", ...)
 - char value = "a";
 - char* values = "bunch of characters";
- bool - true or false
 - bool_truevariable = true;
 - bool_falsevariable = false;

A few basics (bare with me):



One step beyond standard variables:

- `std::string` - class that contains characters + some useful functions and operators.
 - `std::string variable("string of characters");`
- `std::vector` - a dynamically allocated array of a given type.
 - `std::vector<double> vectorofdoubles;`
 - `vectorofdoubles.push_back(3.14);`
- Arrays (`[]`) - a static list of a variable type.
 - `double variable[5];`
 - `Variable[0] = 3.14;`

A few basics (bare with me):



Operators:

- Addition - “+”
 - `double Sumvariable = number1 + number2;`
- Subtraction - “-”
 - `float Subvariable = number1 - number2;`
- Modulus - “%”
 - `Int modulus = 5%3`
- Greater and Less Than - “<”, “>”, “>=”, “<=”
 - `bool greater_than = number1 > number2`
- Is equal - “==”
 - `bool isequal = number1 == number2`
- Not equal - “!=”
 - `Bool notequal = number1 != number2`
- And - “and” (&&)
 - `bool both = number1 and number2`
- Or - “or” (||)
 - `bool either = number1 or number2`

Commenting Code

- Commenting code is very important!!!
- Never assume that it is obvious.
- If you have put something into your code that is incorrect that you know you need to come back and fix, comment it! Even you will forget about it if it compiles without the fix.
- Single line comments can be added with “//”
- Multiple line comments can be added using “/*” to start the comment block and “*/” to end it.

```
//This is a single line comment.  
double a_variable;  
  
/*  
This is a comment that  
is multiple lines. So  
much information!  
*/  
void some_function();
```

A few basics (bare with me):

- Functions are something that performs a task. Functions are useful for not having to rewrite the same code over and over, just define a function and call it when needed.

```
void myvoidfunction(){
    int this_is_a_useless_function = 0;
}

int give_me_an_int(){
    return 6;
}

double give_me_a_double(){
    return 3.14;
}

void switch_these_numbers_but_not_really(double num1, double num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}

void switch_these_numbers_actually(double &num1, double &num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}
```


A few basics (bare with me):

- Functions are something that performs a task. Functions are useful for not having to rewrite the same code over and over, just define a function and call it when needed.

```
void myvoidfunction(){
    int this_is_a_useless_function = 0;
}

int give_me_an_int(){
    return 6;
}

double give_me_a_double(){
    return 3.14;
}

void switch_these_numbers_but_not_really(double num1, double num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}

void switch_these_numbers_actually(double &num1, double &num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}
```

A function that does nothing.

A few basics (bare with me):

- Functions are something that performs a task. Functions are useful for not having to rewrite the same code over and over, just define a function and call it when needed.

```
void myvoidfunction(){
    int this_is_a_useless_function = 0;
}

int give_me_an_int(){
    return 6;
}

double give_me_a_double(){
    return 3.14;
}

void switch_these_numbers_but_not_realy(double num1,double num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}

void switch_these_numbers_actually(double &num1,double &num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}
```

Returns an integer of 6..

A few basics (bare with me):

- Functions are something that performs a task. Functions are useful for not having to rewrite the same code over and over, just define a function and call it when needed.

```
void myvoidfunction(){
    int this_is_a_useless_function = 0;
}

int give_me_an_int(){
    return 6;
}

double give_me_a_double(){
    return 3.14;
}

void switch_these_numbers_but_not_really(double num1, double num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}

void switch_these_numbers_actually(double &num1, double &num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}
```

← Returns an double of 3.14.

A few basics (bare with me):

- Functions are something that performs a task. Functions are useful for not having to rewrite the same code over and over, just define a function and call it when needed.

```
void myvoidfunction(){
    int this_is_a_useless_function = 0;
}

int give_me_an_int(){
    return 6;
}

double give_me_a_double(){
    return 3.14;
}

void switch_these_numbers_but_not_realy(double num1, double num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}

void switch_these_numbers_actually(double &num1, double &num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}
```

← Takes in two numbers and switches their values but this is lost as soon as the function exits. The two variables passed would still have the same values.

A few basics (bare with me):

- Functions are something that performs a task. Functions are useful for not having to rewrite the same code over and over, just define a function and call it when needed.

```
void myvoidfunction(){
    int this_is_a_useless_function = 0;
}

int give_me_an_int(){
    return 6;
}

double give_me_a_double(){
    return 3.14;
}

void switch_these_numbers_but_not_realy(double num1, double num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}

void switch_these_numbers_actually(double &num1, double &num2){
    double temp = num2;
    num2 = num1;
    num1 = temp;
}
```

This function truly takes the variables and switches the values because we passed the address of the variable instead of making a copy of them. .

A few basics (bare with me):



- Scope is the region of code that a variable is defined.
- In c++ the scope is defined by “{}” brackets.

```
int variable1 = 0;
{
    int variable2 = 1; // variable2 only exists within brackets
    variable1 = variable2; //variable1 exists in newly created brackets
}
//variable 1 exists and equals 1 but variable2 does not exist anymore.
```

Namespace

- With the large numbers of libraries and codes out there, it is impossible to have unique names for every function that you might have included in your programs.
- If you need two different math libraries you may end up with many sin and cos functions!
- Defining a namespace allows for functions with the same name to coexist in separate name spaces.
- The standard library namespace “std” is one of the most common namespaces you will see.

```
1 namespace mynamespace
2 {
3     int sin = 0;
4     double cos = 0.0;
5
6     double printf(char* message){
7         std::cout << message << std::endl;
8     }
9 }
10
11 mynamespace::sin = 4;
12 mynamespace::cos = 4.82;
13 mynamespace::printf("print me a message");
14
15 using namespace mynamespace;
16
17 sin = 4;
18 cos = 4.82;
19 printf("print me a message");
```

Made a namespace and defined some thing.

Namespace

- With the large numbers of libraries and codes out there, it is impossible to have unique names for every function that you might have included in your programs.
- If you need two different math libraries you may end up with many sin and cos functions!
- Defining a namespace allows for functions with the same name to coexist in separate name spaces.
- The standard library namespace “std” is one of the most common namespaces you will see.

```
1 namespace mynamespace
2 {
3     int sin = 0;
4     double cos = 0.0;
5
6     double printf(char* message){
7         std::cout << message << std::endl;
8     }
9 }
10
11 mynamespace::sin = 4;
12 mynamespace::cos = 4.82;
13 mynamespace::printf("print me a message");
14
15 using namespace mynamespace;
16
17 sin = 4;
18 cos = 4.82;
19 printf("print me a message");
```

Can call my variables under the namespace.

Namespace

- With the large numbers of libraries and codes out there, it is impossible to have unique names for every function that you might have included in your programs.
- If you need two different math libraries you may end up with many sin and cos functions!
- Defining a namespace allows for functions with the same name to coexist in separate name spaces.
- The standard library namespace “std” is one of the most common namespaces you will see.

```
1 namespace mynamespace
2 {
3     int sin = 0;
4     double cos = 0.0;
5
6     double printf(char* message){
7         std::cout << message << std::endl;
8     }
9 }
10
11 mynamespace::sin = 4;
12 mynamespace::cos = 4.82;
13 mynamespace::printf("print me a message");
14
15 using namespace mynamespace;
16
17 sin = 4;
18 cos = 4.82;
19 printf("print me a message");
```

I can all all thing from this namespace into my standard space and drop the mynamespace::

Namespace

- With the large numbers of libraries and codes out there, it is impossible to have unique names for every function that you might have included in your programs.
- If you need two different math libraries you may end up with many sin and cos functions!
- Defining a namespace allows for functions with the same name to coexist in separate name spaces.
- The standard library namespace “std” is one of the most common namespaces you will see.

```
1 namespace mynamespace
2 {
3     int sin = 0;
4     double cos = 0.0;
5
6     double printf(char* message){
7         std::cout << message << std::endl;
8     }
9 }
10
11 mynamespace::sin = 4;
12 mynamespace::cos = 4.82;
13 mynamespace::printf("print me a message");
14
15 using namespace mynamespace;
16
17 sin = 4;
18 cos = 4.82;
19 printf("print me a message");
```

This can be dangerous as it causes use to have overlapping names!!!

Basic syntax in c++:

```
helloworld.cpp x
1  #include <stdio.h>
2
3  int main(){
4      |
5      |   printf("Yup, another hello world....");
6      |
7      |   return 0;
8      |
9      }
```

Basic syntax in c++:

```
helloworld.cpp x
1 #include <stdio.h>
2
3 int main(){
4     printf("Yup, another hello world....");
5
6     return 0;
7 }
8
9
```

Include code from other files.

- “<>” usually used for installed library headers in standard install location
- “Headerfile.h” used for our local headers or headers that we are including from non standard location

Basic syntax in c++:

```
helloworld.cpp x
1  #include <stdio.h>
2
3  int main(){
4      printf("Yup, another hello world....");
5
6      return 0;
7  }
8
9
```

Define main program function, all programs have a main and usually are defined as an integer. The return value tells the computer if the program completes properly (returns 0).

Basic syntax in c++:

```
helloworld.cpp x
1  #include <stdio.h>
2
3  int main(){
4      printf("Yup, another hello world...");
5
6      return 0;
7  }
8
9
```

Use function printf which is defined in stdio.h, this prints the passed char* to the terminal.

Basic syntax in c++:

```
helloworld.cpp x
1 #include <stdio.h>
2
3 int main(){
4     printf("Yup, another hello world....");
5
6     return 0;
7 }
8
9
```

Return a value for the completion of the main function.

Basic syntax in c++:

```
helloworld.cpp x
1 #include <stdio.h>
2
3 int main(){
4     printf("Yup, another hello world....");
5
6     return 0;
7 }
8
9
```

← Marks end of main function.

Compiling Our first Program:

- C++ code needs to be compiled before it can be run.
- Compilers take the code that we wrote and translates it into machine code.

```
$$ g++ -o helloworld helloworld.cpp
```



The command `g++ -o helloworld helloworld.cpp` is shown with three colored boxes: a green box around `g++`, a yellow box around `-o helloworld`, and a red box around `helloworld.cpp`. A green arrow points from the label "Compiler program" to the green box. A yellow arrow points from the label "Name of program to be made." to the yellow box. A red arrow points from the label "File with code." to the red box.

```
$$ ./helloworld
```

(“./” tells computer that helloworld is located in our current directory.)

If else statements



- **If** statements are used to carry out code under a provided condition.
- They can be paired with following conditions using **else if**.
- **else** catches any other condition.

```
if(number<3.14){
    Number += 0.1;
}else if(number > 3.14){
    Number -= 0.1;
}else {
    //do nothing because number is perfect.
}
```

Loops



- There are two main loops that we use, for and while

for:

```
for("initial condition", "continue condition", "end of loop operator")
```

```
for(int i=0; i<5; ++i) \\ in this case the variable i only exist within {} of for loop.
```

```
double number = 0.54;
```

```
for(; number<=3.14; number += 0.1)
```

while:

```
while("continue condition")
```

```
while(number<=3.14)
```

- In general the loop operations are put within {}; however, for the case of a single line operation, the brackets are not needed.

```
for(int i=0; i<5; ++i){  
    printf("%d\n",i);  
}
```

```
for(int i=0; i<5; ++i)  
    printf("%d\n",i);
```

Loop disruption



- Sometimes you might want to skip over a loop or prematurely stop it entirely.
- The continue command will stop the current iteration of a loop and continue with the next iteration.
- The break command will leave the loop entirely.

```
int sum = 0;
for(int i=1=0; i<10; ++i){
    if(i%2 == 0) continue;
    If (i>5) break;
    sum += i;
}
```

Helloworld 2.0

```
helloworld_input.cpp x
1  #include <iostream>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv){
5
6      if(argc < 2){
7          std::cout << "No arguments given because first argument is program name." << std::endl;
8          return 0;
9      }
10
11     int nprints = atoi(argv[1]); //change char* into an integer
12
13     for(int i=0; i<nprints; ++i){
14         if(i<nprints-1){
15             std::cout << "Print me baby one more time." << std::endl;
16         }else{
17             std::cout << "Last print." << std::endl;
18         }
19     }
20
21     return 0;
22 }
23
```

Header for cout
Header for atoi

Helloworld 2.0

```
helloworld_input.cpp x
1  #include <iostream>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv){
5
6      if(argc < 2){
7          std::cout << "No arguments given because first argument is program name." << std::endl;
8          return 0;
9      }
10
11     int nprints = atoi(argv[1]); //change char* into an integer
12
13     for(int i=0; i<nprints; ++i){
14         if(i<nprints-1){
15             std::cout << "Print me baby one more time." << std::endl;
16         }else{
17             std::cout << "Last print." << std::endl;
18         }
19     }
20
21     return 0;
22 }
23
```

Add command line inputs
For the program.

Argc is the number of
arguments, the program
name is considered the
first argument.

argv is a pointer to an
array of char*'s for each
input string.

Helloworld 2.0

```
helloworld_input.cpp x
1  #include <iostream>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv){
5
6      if(argc < 2){
7          std::cout << "No arguments given because first argument is program name." << std::endl;
8          return 0;
9      }
10
11     int nprints = atoi(argv[1]); //change char* into an integer
12
13     for(int i=0; i<nprints; ++i){
14         if(i<nprints-1){
15             std::cout << "Print me baby one more time." << std::endl;
16         }else{
17             std::cout << "Last print." << std::endl;
18         }
19     }
20
21     return 0;
22 }
23
```

Check if we don't have additional arguments, then close program by returning within main.

Helloworld 2.0

```
helloworld_input.cpp x
1  #include <iostream>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv){
5
6      if(argc < 2){
7          std::cout << "No arguments given because first argument is program name." << std::endl;
8          return 0;
9      }
10
11     int nprints = atoi(argv[1]); //change char* into an integer
12
13     for(int i=0; i<nprints; ++i){
14         if(i<nprints-1){
15             std::cout << "Print me baby one more time." << std::endl;
16         }else{
17             std::cout << "Last print." << std::endl;
18         }
19     }
20
21     return 0;
22 }
23
```

Convert the char* in argv to an integer value. If we wanted a float we would use atof.

Helloworld 2.0

```
helloworld_input.cpp x
1  #include <iostream>
2  #include <stdlib.h>
3
4  int main(int argc, char** argv){
5
6      if(argc < 2){
7          std::cout << "No arguments given because first argument is program name." << std::endl;
8          return 0;
9      }
10
11     int nprints = atoi(argv[1]); //change char* into an integer
12
13     for(int i=0; i<nprints; ++i){
14         if(i<nprints-1){
15             std::cout << "Print me baby one more time." << std::endl;
16         }else{
17             std::cout << "Last print." << std::endl;
18         }
19     }
20
21     return 0;
22 }
23
```

Our simple for loop and if else statement.

Make a header.



- Headers are ways to split up the code into manageable pieces, improving readability.
- The header must declare the functions and variables but the actual definitions and assignments can be put into a separate source file.
- In large pieces of code, it is important to insure against defining the same functions more than once. This is done with the `#define` call.

Helloworld with a header.

```
helloheader.h x
1  #ifndef _HELLOHEADER_
2  #define _HELLOHEADER_
3
4  #include <iostream>
5
6  void helloworldprint(){
7      std::cout << "Hello world :)"<<std::endl;
8  }
9
10 #endif
```

This makes sure that the header is only read in once. If multiple files in our project needs this header then then compiler can get confused.

```
helloworld_withheader.cpp x
1  #include "helloheader.h"
2
3  int main(){
4
5      helloworldprint();
6
7      return 0;
8
9  }
```

Hello world with a header.

```
helloheader.h x
1  #ifndef _HELLOHEADER_
2  #define _HELLOHEADER_
3
4  #include <iostream>
5
6  void helloworldprint(){
7      std::cout << "Hello world :)"<<std::endl;
8  }
9
10 #endif
```

A simple function that doesn't take in any variables and doesn't return anything (void).

```
helloworld_withheader.cpp x
1  #include "helloheader.h"
2
3  int main(){
4
5      helloworldprint();
6
7      return 0;
8
9  }
```

Hello world with a header.

```
helloheader.h x
1  #ifndef _HELLOHEADER_
2  #define _HELLOHEADER_
3
4  #include <iostream>
5
6  void helloworldprint(){
7      std::cout << "Hello world :)"<<std::endl;
8  }
9
10 #endif
```

```
helloworld_withheader.cpp x
1  #include "helloheader.h"
2
3  int main(){
4
5      helloworldprint();
6
7      return 0;
8
9  }
```

Include our header.

Hello world with a header.

```
helloheader.h x
1  #ifndef _HELLOHEADER_
2  #define _HELLOHEADER_
3
4  #include <iostream>
5
6  void helloworldprint(){
7      std::cout << "Hello world :)"<<std::endl;
8  }
9
10 #endif
```

```
helloworld_withheader.cpp x
1  #include "helloheader.h"
2
3  int main(){
4
5      helloworldprint();
6
7      return 0;
8
9  }
```

Call the function.

Compiling



- As projects become more complex, it is less practical to type the compile commands into the terminal everytime.
- There are several ways to make compiling easier, the two that we will quickly discuss are GnuMake and CMake.
- Most projects are switching to CMake.

GNU Make

- A very BASH like compiling script.
- White space (indentation is important)
- Just type make while in directory with the Makefile to build.
- Keeps track of files that have been changed since last compile and will skip things that have not changed.

Target name



```
Makefile
1 helloworld: helloworld.cpp
2   g++ -o $@ $^
3
```

Prerequisites

Tab space

Same compile command but with some short cuts.
\$@ inputs the target name and \$^ inputs the list of prerequisites.

More references:

https://www.gnu.org/software/make/manual/html_node/Overview.html

<https://devhints.io/makefile>

CMake

- Now the standard for any new projects.
- You will find that sometimes CMake is only used to make a Makefile, and then GNU Make is used for actual building.

```
CMakeLists.txt x
1 cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
2 project(helloworldproject)
3
4 add_executable(helloworld src/helloworld.cpp)
5 include_directories(${PROJECT_SOURCE_DIR}/include)
6
7 #-----
8 # Install the executable to 'bin' directory under CMAKE_INSTALL_PREFIX
9 #
10
11 install(TARGETS helloworld DESTINATION bin)
12
```

Make a project

CMake

- Now the standard for any new projects.
- You will find that sometimes CMake is only used to make a Makefile, and then GNU Make is used for actual building.

```
CMakeLists.txt x
1 cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
2 project(helloworldproject)
3
4 add_executable(helloworld src/helloworld.cpp)
5 include_directories(${PROJECT_SOURCE_DIR}/include)
6
7 #-----
8 # Install the executable to 'bin' directory under CMAKE_INSTALL_PREFIX
9 #
10
11 install(TARGETS helloworld DESTINATION bin)
12
```

Add an executable program to project.

CMake

- Now the standard for any new projects.
- You will find that sometimes CMake is only used to make a Makefile, and then GNU Make is used for actual building.

```
CMakeLists.txt x
1 cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
2 project(helloworldproject)
3
4 add_executable(helloworld src/helloworld.cpp)
5 include_directories(${PROJECT_SOURCE_DIR}/include)
6
7 #-----
8 # Install the executable to 'bin' directory under CMAKE_INSTALL_PREFIX
9 #
10
11 install(TARGETS helloworld DESTINATION bin)
12
```

Tell it where to find headers that are not in standard locations.

CMake

- Now the standard for any new projects.
- You will find that sometimes CMake is only used to make a Makefile, and then GNU Make is used for actual building.

```
CMakeLists.txt x
1  cmake_minimum_required(VERSION 2.6 FATAL_ERROR)
2  project(helloworldproject)
3
4  add_executable(helloworld src/helloworld.cpp)
5  include_directories(${PROJECT_SOURCE_DIR}/include)
6
7  #-----
8  # Install the executable to 'bin' directory under CMAKE_INSTALL_PREFIX
9  #
10
11  install(TARGETS helloworld DESTINATION bin)
12
```

Install instructions if needed.

More info on CMake

<https://cmake.org/cmake/help/latest/guide/tutorial/index.html>

Using CMake



- `$mkdir build` (make a directory for project to build in)
- `$cd build` (enter build directory)
- `$cmake ../` (build compile instructions, ../ assumes you made the build in project directory)
- `$cmake --build ./` (build the project in current directory)

Classes



- Classes are the updated version of structures in C.
- They are a collection of functions and variables.
- Can define operators to act on them in specific ways.
- Classes can inherit from other classes and can be used in place of their base class.

Classes

```
helloclass.h x
1 #ifndef _hellolib_
2 #define _hellolib_
3
4 class hello
5 {
6     public:
7     hello(char* _hellomessage);
8     ~hello();
9     void sayhello();
10    void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11    char* GetHelloMessage(){return hellomessage;}
12    char* publicmessage;
13
14    private:
15    char* hellomessage;
16 };
17 #endif
```

Class name, if we wanted to inherit from another class then we would add ":" then all parent classes.

```
helloclass.cpp x
1 #include "helloclass.h"
2 #include <stdio.h>
3
4 hello::hello(char* _hellomessage){
5     hellomessage = _hellomessage;
6 }
7
8 hello::~~hello(){
9 }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Classes

```
helloclass.h x
1  #ifndef _hellolib_
2  #define _hellolib_
3
4  class hello
5  {
6      public:
7      hello(char* _hellomessage);
8      ~hello();
9      void sayhello();
10     void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11     char* GetHelloMessage(){return hellomessage;}
12     char* publicmessage;
13
14     private:
15     char* hellomessage;
16 };
17 #endif
```

Anything in public space is callable from outside the class.

```
helloclass.cpp x
1  #include "helloclass.h"
2  #include <stdio.h>
3
4  hello::hello(char* _hellomessage){
5      hellomessage = _hellomessage;
6  }
7
8  hello::~~hello(){
9  }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```


Classes

```
helloclass.h x
1  #ifndef _hellolib_
2  #define _hellolib_
3
4  class hello
5  {
6      public:
7      hello(char* _hellomessage);
8      ~hello();
9      void sayhello();
10     void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11     char* GetHelloMessage(){return hellomessage;}
12     char* publicmessage;
13
14     private: ←
15     char* hellomessage;
16 };
17 #endif
```

```
helloclass.cpp x
1  #include "helloclass.h"
2  #include <stdio.h>
3
4  hello::hello(char* _hellomessage){
5      hellomessage = _hellomessage;
6  }
7
8  hello::~~hello(){
9  }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Anything in private space is protected and only class functions can directly access.

**Can also have protected variables which are like private variables but can be accessed by classes that inherit from parent class.

Classes

```
helloclass.h x
1  #ifndef _hellolib_
2  #define _hellolib_
3
4  class hello
5  {
6      public:
7      hello(char* _hellomessage);
8      ~hello();
9      void sayhello();
10     void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11     char* GetHelloMessage(){return hellomessage;}
12     char* publicmessage;
13
14     private:
15     char* hellomessage;
16 };
17 #endif
```

Constructor, sets up initial state of class. Can have multiple constructors.

```
helloclass.cpp x
1  #include "helloclass.h"
2  #include <stdio.h>
3
4  hello::hello(char* _hellomessage){
5      hellomessage = _hellomessage;
6  }
7
8  hello::~~hello(){
9  }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Classes

```
helloclass.h x
1 #ifndef _hellolib_
2 #define _hellolib_
3
4 class hello
5 {
6     public:
7     hello(char* _hellomessage);
8     ~hello();
9     void sayhello();
10    void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11    char* GetHelloMessage(){return hellomessage;}
12    char* publicmessage;
13
14    private:
15    char* hellomessage;
16 };
17 #endif
```

Destructor, how to nicely kill class (delete pointes).

```
helloclass.cpp x
1 #include "helloclass.h"
2 #include <stdio.h>
3
4 hello::hello(char* _hellomessage){
5     hellomessage = _hellomessage;
6 }
7
8 hello::~~hello(){
9 }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Classes

```
helloclass.h x
1 #ifndef _hellolib_
2 #define _hellolib_
3
4 class hello
5 {
6     public:
7     hello(char* _hellomessage);
8     ~hello();
9     void sayhello();
10    void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11    char* GetHelloMessage(){return hellomessage;}
12    char* publicmessage;
13
14    private:
15    char* hellomessage;
16 };
17 #endif
```

Useful function

```
helloclass.cpp x
1 #include "helloclass.h"
2 #include <stdio.h>
3
4 hello::hello(char* _hellomessage){
5     hellomessage = _hellomessage;
6 }
7
8 hello::~~hello(){
9 }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Classes

```
helloclass.h x
1 #ifndef _hellolib_
2 #define _hellolib_
3
4 class hello
5 {
6     public:
7     hello(char* _hellomessage);
8     ~hello();
9     void sayhello();
10    void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11    char* GetHelloMessage(){return hellomessage;}
12    char* publicmessage;
13
14    private:
15    char* hellomessage;
16 };
17 #endif
```

```
helloclass.cpp x
1 #include "helloclass.h"
2 #include <stdio.h>
3
4 hello::hello(char* _hellomessage){
5     hellomessage = _hellomessage;
6 }
7
8 hello::~~hello(){
9 }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Getter and setter for private variable.

Classes

```
helloclass.h x
1 #ifndef _hellolib_
2 #define _hellolib_
3
4 class hello
5 {
6     public:
7     hello(char* _hellomessage);
8     ~hello();
9     void sayhello();
10    void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11    char* GetHelloMessage(){return hellomessage;}
12    char* publicmessage;
13
14    private:
15    char* hellomessage;
16 };
17 #endif
```

Public variable

```
helloclass.cpp x
1 #include "helloclass.h"
2 #include <stdio.h>
3
4 hello::hello(char* _hellomessage){
5     hellomessage = _hellomessage;
6 }
7
8 hello::~~hello(){
9 }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Classes

```
helloclass.h x
1 #ifndef _hellolib_
2 #define _hellolib_
3
4 class hello
5 {
6     public:
7     hello(char* _hellomessage);
8     ~hello();
9     void sayhello();
10    void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11    char* GetHelloMessage(){return hellomessage;}
12    char* publicmessage;
13
14    private:
15    char* hellomessage;
16 };
17 #endif
```

Private variable

```
helloclass.cpp x
1 #include "helloclass.h"
2 #include <stdio.h>
3
4 hello::hello(char* _hellomessage){
5     hellomessage = _hellomessage;
6 }
7
8 hello::~~hello(){
9 }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Classes

```
helloclass.h x
1 #ifndef _hellolib_
2 #define _hellolib_
3
4 class hello
5 {
6     public:
7     hello(char* _hellomessage);
8     ~hello();
9     void sayhello();
10    void SetHelloMessage(char* _hellomessage){hellomessage = _hellomessage;}
11    char* GetHelloMessage(){return hellomessage;}
12    char* publicmessage;
13
14    private:
15    char* hellomessage;
16 };
17 #endif
```

```
helloclass.cpp x
1 #include "helloclass.h"
2 #include <stdio.h>
3
4 hello::hello(char* _hellomessage){
5     hellomessage = _hellomessage;
6 }
7
8 hello::~~hello(){
9 }
10
11 void hello::sayhello(){
12     printf("%s\n",hellomessage);
13 }
14
```

Source file to define functions in header.

Libraries



- Groups of useful code can be grouped into a library to make it easier to use.
- With a library, all you need is the library file and the headers to use the code, the source files are not needed.
- There are several types of libraries static, dynamic, shared...
- CMake makes it really easy to build a library and add it to your project.
- Linking a library manually link libcool with “-lcool” for libraries in standard location.
- Use “-L path/to/libcool.so” if not in standard location.

Make library for hello class



```
CMakeLists.txt x
1 cmake_minimum_required(VERSION 3.6)
2 project(hello1ib)
3
4 include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include)
5
6 add_library(hello1ib SHARED ${CMAKE_CURRENT_SOURCE_DIR}/src/helloclass.cpp)
7
8 list(APPEND CMAKE_PREFIX_PATH ${CMAKE_CURRENT_BINARY_DIR})
9
10 add_executable(helloworld main.cpp)
11 include_directories(${PROJECT_SOURCE_DIR}/include)
12 target_link_libraries(helloworld PRIVATE hello1ib)
13
```

← Make library

Make library for hello class



```
CMakeLists.txt x
1 cmake_minimum_required(VERSION 3.6)
2 project(hello1ib)
3
4 include_directories(${CMAKE_CURRENT_SOURCE_DIR}/include)
5
6 add_library(hello1ib SHARED ${CMAKE_CURRENT_SOURCE_DIR}/src/hello1class.cpp)
7
8 list(APPEND CMAKE_PREFIX_PATH ${CMAKE_CURRENT_BINARY_DIR})
9
10 add_executable(hello1world main.cpp)
11 include_directories(${PROJECT_SOURCE_DIR}/include)
12 target_link_libraries(hello1world PRIVATE hello1lib)
13
```

Link it to our program.

For libraries that are not built by us, we would either use the the FindPackage CMake function to find the library and then use the correct library name specified in it's config file or we can also manually put in the GNU style -L...

Pointers



- So far we have worked with plain variables; however to make programs more efficient, c++ has the ability to define variables by their address.
- Pointers allow object to be more easily shared between functions and can speed up processing since the memory of a large object is not being copied into the function, only the address.
- Pointers are declared using the * character, (yes char* is a pointer).
- Need to remember that when creating a pointer, it creates 2 variables. One is a variable that stores the address for memory and the other is the actual object itself. When we are done with the object we must manually clear that memory or else it will persist until the program ends. This can create memory leaks.

Put it together into a program:

```
main.cpp x
1  #include <iostream>
2  #include "helloclass.h"
3
4  int main(int argc, char** argv){
5
6      hello myhello = hello("#freebritney");
7      myhello.publicmessage = "#AlwaysMoneyInTheBananaStand";
8      myhello.sayhello();
9      std::cout << myhello.publicmessage << std::endl;
10     myhello.SetHelloMessage("#gonecrazy");
11     myhello.sayhello();
12
13     hello* pointerhello = new hello("#imaduck");
14     pointerhello->publicmessage = "#uaduck";
15     pointerhello->sayhello();
16     std::cout << pointerhello->publicmessage << std::endl;
17     pointerhello->SetHelloMessage("#weallducks");
18     pointerhello->sayhello();
19
20     delete pointerhello;
21
22     pointerhello = &myhello;
23     pointerhello->sayhello();
24
25     return 0;
26 }
```

Include our hello class.

Put it together into a program:

```
main.cpp x
1  #include <iostream>
2  #include "helloclass.h"
3
4  int main(int argc, char** argv){
5
6      hello myhello = hello("#freebritney");
7      myhello.publicmessage = "#AlwaysMoneyInTheBananaStand";
8      myhello.sayhello();
9      std::cout << myhello.publicmessage << std::endl;
10     myhello.SetHelloMessage("#gonecrazy");
11     myhello.sayhello();
12
13     hello* pointerhello = new hello("#imaduck");
14     pointerhello->publicmessage = "#uaduck";
15     pointerhello->sayhello();
16     std::cout << pointerhello->publicmessage << std::endl;
17     pointerhello->SetHelloMessage("#weallducks");
18     pointerhello->sayhello();
19
20     delete pointerhello;
21
22     pointerhello = &myhello;
23     pointerhello->sayhello();
24
25     return 0;
26 }
```

Define a hello object.

Put it together into a program:

```
main.cpp x
1  #include <iostream>
2  #include "helloclass.h"
3
4  int main(int argc, char** argv){
5
6      hello myhello = hello("#freebritney");
7      myhello.publicmessage = "#AlwaysMoneyInTheBananaStand";
8      myhello.sayhello();
9      std::cout << myhello.publicmessage << std::endl;
10     myhello.SetHelloMessage("#gonecrazy");
11     myhello.sayhello();
12
13     hello* pointerhello = new hello("#imaduck");
14     pointerhello->publicmessage = "#uaduck";
15     pointerhello->sayhello();
16     std::cout << pointerhello->publicmessage << std::endl;
17     pointerhello->SetHelloMessage("#weallducks");
18     pointerhello->sayhello();
19
20     delete pointerhello;
21
22     pointerhello = &myhello;
23     pointerhello->sayhello();
24
25     return 0;
26 }
```

We can directly access our public variable and functions.

Put it together into a program:

```
main.cpp x
1  #include <iostream>
2  #include "helloclass.h"
3
4  int main(int argc, char** argv){
5
6      hello myhello = hello("#freebritney");
7      myhello.publicmessage = "#AlwaysMoneyInTheBananaStand";
8      myhello.sayhello();
9      std::cout << myhello.publicmessage << std::endl;
10     myhello.SetHelloMessage("#gonecrazy");
11     myhello.sayhello();
12
13     hello* pointerhello = new hello("#imaduck");
14     pointerhello->publicmessage = "#uaduck";
15     pointerhello->sayhello();
16     std::cout << pointerhello->publicmessage << std::endl;
17     pointerhello->SetHelloMessage("#weallducks");
18     pointerhello->sayhello();
19
20     delete pointerhello;
21
22     pointerhello = &myhello;
23     pointerhello->sayhello();
24
25     return 0;
26 }
```

Private variable must be accessed through public setter.

Put it together into a program:

```
main.cpp x
1  #include <iostream>
2  #include "helloclass.h"
3
4  int main(int argc, char** argv){
5
6      hello myhello = hello("#freebritney");
7      myhello.publicmessage = "#AlwaysMoneyInTheBananaStand";
8      myhello.sayhello();
9      std::cout << myhello.publicmessage << std::endl;
10     myhello.SetHelloMessage("#gonecrazy");
11     myhello.sayhello();
12
13     hello* pointerhello = new hello("#imaduck");
14     pointerhello->publicmessage = "#uaduck";
15     pointerhello->sayhello();
16     std::cout << pointerhello->publicmessage << std::endl;
17     pointerhello->SetHelloMessage("#weallducks");
18     pointerhello->sayhello();
19
20     delete pointerhello;
21
22     pointerhello = &myhello;
23     pointerhello->sayhello();
24
25     return 0;
26 }
```

Declare a pointer to a new hello object.

Put it together into a program:

```
main.cpp x
1  #include <iostream>
2  #include "helloclass.h"
3
4  int main(int argc, char** argv){
5
6      hello myhello = hello("#freebritney");
7      myhello.publicmessage = "#AlwaysMoneyInTheBananaStand";
8      myhello.sayhello();
9      std::cout << myhello.publicmessage << std::endl;
10     myhello.SetHelloMessage("#gonecrazy");
11     myhello.sayhello();
12
13     hello* pointerhello = new hello("#imaduck");
14     pointerhello->publicmessage = "#uaduck";
15     pointerhello->sayhello();
16     std::cout << pointerhello->publicmessage << std::endl;
17     pointerhello->SetHelloMessage("#weallducks");
18     pointerhello->sayhello();
19
20     delete pointerhello;
21
22     pointerhello = &myhello;
23     pointerhello->sayhello();
24
25     return 0;
26 }
```

Now, instead of "." to access functions and variables, "->" is used.

Put it together into a program:

```
main.cpp x
1  #include <iostream>
2  #include "helloclass.h"
3
4  int main(int argc, char** argv){
5
6      hello myhello = hello("#freebritney");
7      myhello.publicmessage = "#AlwaysMoneyInTheBananaStand";
8      myhello.sayhello();
9      std::cout << myhello.publicmessage << std::endl;
10     myhello.SetHelloMessage("#gonecrazy");
11     myhello.sayhello();
12
13     hello* pointerhello = new hello("#imaduck");
14     pointerhello->publicmessage = "#uaduck";
15     pointerhello->sayhello();
16     std::cout << pointerhello->publicmessage << std::endl;
17     pointerhello->SetHelloMessage("#weallducks");
18     pointerhello->sayhello();
19
20     delete pointerhello;
21
22     pointerhello = &myhello;
23     pointerhello->sayhello();
24
25     return 0;
26 }
```

We no longer need the object that our pointer is pointing to, then we can delete it.

Put it together into a program:

```
main.cpp x
1  #include <iostream>
2  #include "helloclass.h"
3
4  int main(int argc, char** argv){
5
6      hello myhello = hello("#freebritney");
7      myhello.publicmessage = "#AlwaysMoneyInTheBananaStand";
8      myhello.sayhello();
9      std::cout << myhello.publicmessage << std::endl;
10     myhello.SetHelloMessage("#gonecrazy");
11     myhello.sayhello();
12
13     hello* pointerhello = new hello("#imaduck");
14     pointerhello->publicmessage = "#uaduck";
15     pointerhello->sayhello();
16     std::cout << pointerhello->publicmessage << std::endl;
17     pointerhello->SetHelloMessage("#weallducks");
18     pointerhello->sayhello();
19
20     delete pointerhello;
21
22     pointerhello = &myhello;
23     pointerhello->sayhello();
24
25     return 0;
26 }
```

We can access the address of a variable with the & operator. You can also directly access the object that a pointer is pointing to using *. (*pointerhello).sayhello();

Vectors

- Vectors are a widely used variable type in physics.
- The vector is actually a template class that can take on any class that we want.
- Essentially allows us to have a variable length array and can even sort the order and rearrange the contents.

```
#include <vector>

std::vector<int> myvectorofints;

for(int i=100; i>0; --i)
    myvectorofints.push_back(i);

std::sort(myvectorofints.begin(), myvectorofints.end());

int n_elements_in_vector = myvectorofints.size();
for(int i=0; i<n_elements_in_vector; ++i) printf("%d\n",myvectorofints[i]);

int array[] = {10,9,8,7,6,5,4,3,2,1,0};
int n_elements_in_array = sizeof(array)/sizeof(array[0]);
```

Vectors

- Vectors are a widely used variable type in physics.
- The vector is actually a template class that can take on any class that we want.
- Essentially allows us to have a variable length array and can even sort the order and rearrange the contents.

```
#include <vector>

std::vector<int> myvectorofints;

for(int i=100; i>0; --i)
    myvectorofints.push_back(i);

std::sort(myvectorofints.begin(), myvectorofints.end());

int n_elements_in_vector = myvectorofints.size();
for(int i=0; i<n_elements_in_vector; ++i) printf("%d\n",myvectorofints[i]);

int array[] = {10,9,8,7,6,5,4,3,2,1,0};
int n_elements_in_array = sizeof(array)/sizeof(array[0]);
```

← Header for vector class.

Vectors

- Vectors are a widely used variable type in physics.
- The vector is actually a template class that can take on any class that we want.
- Essentially allows us to have a variable length array and can even sort the order and rearrange the contents.

```
#include <vector>

std::vector<int> myvectorofints;

for(int i=100; i>0; --i)
    myvectorofints.push_back(i);

std::sort(myvectorofints.begin(), myvectorofints.end());

int n_elements_in_vector = myvectorofints.size();
for(int i=0; i<n_elements_in_vector; ++i) printf("%d\n",myvectorofints[i]);

int array[] = {10,9,8,7,6,5,4,3,2,1,0};
int n_elements_in_array = sizeof(array)/sizeof(array[0]);
```

Define a vector of ints, can put nearly any class in "<>" and make vector of it.

Vectors

- Vectors are a widely used variable type in physics.
- The vector is actually a template class that can take on any class that we want.
- Essentially allows us to have a variable length array and can even sort the order and rearrange the contents.

```
#include <vector>


std::vector<int> myvectorofints;

for(int i=100; i>0; --i)
    myvectorofints.push_back(i);

std::sort(myvectorofints.begin(), myvectorofints.end());

int n_elements_in_vector = myvectorofints.size();
for(int i=0; i<n_elements_in_vector; ++i) printf("%d\n",myvectorofints[i]);

int array[] = {10,9,8,7,6,5,4,3,2,1,0};
int n_elements_in_array = sizeof(array)/sizeof(array[0]);
```



Add elements to the vector, new elements are added to the end.

Vectors

- Vectors are a widely used variable type in physics.
- The vector is actually a template class that can take on any class that we want.
- Essentially allows us to have a variable length array and can even sort the order and rearrange the contents.

```
#include <vector>

std::vector<int> myvectorofints;

for(int i=100; i>0; --i)
    myvectorofints.push_back(i);

std::sort(myvectorofints.begin(), myvectorofints.end());

int n_elements_in_vector = myvectorofints.size();
for(int i=0; i<n_elements_in_vector; ++i) printf("%d\n",myvectorofints[i]);

int array[] = {10,9,8,7,6,5,4,3,2,1,0};
int n_elements_in_array = sizeof(array)/sizeof(array[0]);
```

Built in sorting function, by default it puts it in order from smallest to biggest. Can be given custom sort functions.

Vectors

- Vectors are a widely used variable type in physics.
- The vector is actually a template class that can take on any class that we want.
- Essentially allows us to have a variable length array and can even sort the order and rearrange the contents.

```
#include <vector>

std::vector<int> myvectorofints;

for(int i=100; i>0; --i)
    myvectorofints.push_back(i);

std::sort(myvectorofints.begin(), myvectorofints.end());

int n_elements_in_vector = myvectorofints.size();
for(int i=0; i<n_elements_in_vector; ++i) printf("%d\n",myvectorofints[i]);

int array[] = {10,9,8,7,6,5,4,3,2,1,0};
int n_elements_in_array = sizeof(array)/sizeof(array[0]);
```

Vector knows how long it is

Vectors

- Vectors are a widely used variable type in physics.
- The vector is actually a template class that can take on any class that we want.
- Essentially allows us to have a variable length array and can even sort the order and rearrange the contents.

```
#include <vector>

std::vector<int> myvectorofints;

for(int i=100; i>0; --i)
    myvectorofints.push_back(i);

std::sort(myvectorofints.begin(), myvectorofints.end());

int n_elements_in_vector = myvectorofints.size();
for(int i=0; i<n_elements_in_vector; ++i) printf("%d\n",myvectorofints[i]);

int array[] = {10,9,8,7,6,5,4,3,2,1,0};
int n_elements_in_array = sizeof(array)/sizeof(array[0]);
```

← Elements are accessed through "[i]" like an array.

Vectors

- Vectors are a widely used variable type in physics.
- The vector is actually a template class that can take on any class that we want.
- Essentially allows us to have a variable length array and can even sort the order and rearrange the contents.

```
#include <vector>

std::vector<int> myvectorofints;

for(int i=100; i>0; --i)
    myvectorofints.push_back(i);

std::sort(myvectorofints.begin(), myvectorofints.end());

int n_elements_in_vector = myvectorofints.size();
for(int i=0; i<n_elements_in_vector; ++i) printf("%d\n",myvectorofints[i]);

int array[] = {10,9,8,7,6,5,4,3,2,1,0};
int n_elements_in_array = sizeof(array)/sizeof(array[0]);
```

In contrast, once arrays are made they are fixed length and do not know their own size, but find out by looking at size of array vs size of one element..

Basic text input and output



- One last basic example.
- We often want to read in data from files and save data to a file.
- We more often use ROOT or other higher end libraries for this but sometimes we have/need a simple CSV file.

Basic text input

```
#include <vector>
#include <string>
#include <cassert>
#include <fstream>
#include <sstream>
#include <iostream>

std::vector<double> GetVectorDouble(char* filename){
    std::ifstream in;
    in.open(filename, std::ifstream::in);
    // Check if object is valid
    if(!in){
        std::cerr << "Cannot open the File : "<<filename<<std::endl;
        return std::vector<double>(0,0.0);
    }
    std::string str;
    std::vector<double> array;
    while(std::getline(in, str)){

        std::replace(str.begin(),str.end(),',',' ');

        std::stringstream ss(str);
        double temp;
        while (ss >> temp){
            array.push_back(temp);
        }
    }
    return array;
}
```

Define and set input file.

Basic text input

```
#include <vector>
#include <string>
#include <cassert>
#include <fstream>
#include <sstream>
#include <iostream>

std::vector<double> GetVectorDouble(char* filename){
    std::ifstream in;
    in.open(filename, std::ifstream::in);
    // Check if object is valid
    if(!in){
        std::cerr << "Cannot open the File : "<<filename<<std::endl;
        return std::vector<double>(0,0.0);
    }
    std::string str;
    std::vector<double> array;
    while(std::getline(in, str)){

        std::replace(str.begin(),str.end(),',',' ');

        std::stringstream ss(str);
        double temp;
        while (ss >> temp){
            array.push_back(temp);
        }
    }
    return array;
}
```

Check that it opened properly.

Basic text input

```
#include <vector>
#include <string>
#include <cassert>
#include <fstream>
#include <sstream>
#include <iostream>

std::vector<double> GetVectorDouble(char* filename){
    std::ifstream in;
    in.open(filename, std::ifstream::in);
    // Check if object is valid
    if(!in){
        std::cerr << "Cannot open the File : "<<filename<<std::endl;
        return std::vector<double>(0,0.0);
    }
    std::string str;
    std::vector<double> array;
    while(std::getline(in, str)){

        std::replace(str.begin(),str.end(),',',' ');

        std::stringstream ss(str);
        double temp;
        while (ss >> temp){
            array.push_back(temp);
        }
    }
    return array;
}
```

Replace commas with white space.

Basic text input

```
#include <vector>
#include <string>
#include <cassert>
#include <fstream>
#include <sstream>
#include <iostream>

std::vector<double> GetVectorDouble(char* filename){
    std::ifstream in;
    in.open(filename, std::ifstream::in);
    // Check if object is valid
    if(!in){
        std::cerr << "Cannot open the File : "<<filename<<std::endl;
        return std::vector<double>(0,0.0);
    }
    std::string str;
    std::vector<double> array;
    while(std::getline(in, str)){

        std::replace(str.begin(),str.end(),',',' ');

        std::stringstream ss(str);
        double temp;
        while (ss >> temp){
            array.push_back(temp);
        }
    }
    return array;
}
```

Read string split by white space as separate numbers and put into a double variable.

Basic text input

```
#include <vector>
#include <string>
#include <cassert>
#include <fstream>
#include <sstream>
#include <iostream>

std::vector<double> GetVectorDouble(char* filename){
    std::ifstream in;
    in.open(filename, std::ifstream::in);
    // Check if object is valid
    if(!in){
        std::cerr << "Cannot open the File : "<<filename<<std::endl;
        return std::vector<double>(0,0.0);
    }
    std::string str;
    std::vector<double> array;
    while(std::getline(in, str)){

        std::replace(str.begin(),str.end(),',',' ');

        std::stringstream ss(str);
        double temp;
        while (ss >> temp){
            array.push_back(temp);
        }
    }
    return array;
}
```

Fill vector.

Basic text input

```
#include <vector>
#include <string>
#include <cassert>
#include <fstream>
#include <sstream>
#include <iostream>

std::vector<double> GetVectorDouble(char* filename){
    std::ifstream in;
    in.open(filename, std::ifstream::in);
    // Check if object is valid
    if(!in){
        std::cerr << "Cannot open the File : "<<filename<<std::endl;
        return std::vector<double>(0,0.0);
    }
    std::string str;
    std::vector<double> array;
    while(std::getline(in, str)){

        std::replace(str.begin(),str.end(),',',' ');

        std::stringstream ss(str);
        double temp;
        while (ss >> temp){
            array.push_back(temp);
        }
    }
    return array;
}
```

Return vector. (Oops, forgot to close file)

Basic text output

```
void SaveVectorDouble(char* filename, std::vector<double> values){
    std::ofstream out;
    out.open(filename, std::ofstream::out);

    int vecsize = values.size();
    for(int i=0; i<vecsize; ++i){
        out << values[i] << ",";
    }

    out.close();
}
```

Receive file name and vector of doubles.

Basic text output

```
void SaveVectorDouble(char* filename, std::vector<double> values){
    std::ofstream out;
    out.open(filename, std::ofstream::out);

    int vecsize = values.size();
    for(int i=0; i<vecsize; ++i){
        out << values[i] << ",";
    }

    out.close();
}
```

Open file for output.

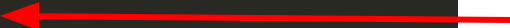
Basic text output



```
void SaveVectorDouble(char* filename, std::vector<double> values){
    std::ofstream out;
    out.open(filename, std::ofstream::out);

    int vecsize = values.size();
    for(int i=0; i<vecsize; ++i){
        out << values[i] << ",";
    }

    out.close();
}
```



Loop over vector and write to file separating with commas.


Basic text output



```
void SaveVectorDouble(char* filename, std::vector<double> values){
    std::ofstream out;
    out.open(filename, std::ofstream::out);

    int vecsize = values.size();
    for(int i=0; i<vecsize; ++i){
        out << values[i] << ",";
    }

    out.close();
}
```



Close file.

Additional Resources



- Just google it, seriously, I do this hourly.
- <https://stackoverflow.com/>
- <https://en.cppreference.com/w/>