C++ Overview

Chapter 1

Note: All commands you type (including the Myro commands listed elsewhere) are essentially C++ commands. Later, in this section we will list those commands that are a part of the C++ language.

Chapter 2

```
#include "FILE NAME"
```

Includes the contents of the file named `FILE NAME` just as though it had been typed at this location in the current file. The file is sought in the same directory as your other C++ programs.

```
void <FUNCTION NAME>(<PARAMETERS>)
{
    <SOMETHING>
    ...
    <SOMETHING>
}
```

Defines a new command/function named `<FUNCTION NAME>`. A function name should always begin with a letter and can be followed by any sequence of letters, numbers, or underscores (_), and not contain any spaces. Try to choose names that appropriately describe the command being defined.

Chapter 3

Values

Values in C++ include numbers (integers or floating point numbers) and strings. Each type of value can be used in an expression by itself or using a combination of operations defined for that type (for example, `+`, `-`, `*`, `/`, `%` for numbers). Strings are considered sequences of characters (or letters).

Names

A name in C++ must begin with either an alphabetic letter (`a-z` or `A-Z`) or the underscore (i.e. _) and can be followed by any sequence of letters, digits, or underscore letters.

```
cin >> <variable>;
```

Waits for the user to type a value compatible with the type of the variable. This value becomes the new value of the variable.

```
#include "Myro.h"
<any other includes>
<function definitions>
int main() {
  connect();
  <do something>
  <do something>
  ...

  disconnect();
} // end of main program
```

This is the basic structure of a robot control program in C++. Without the `#include` line and the `connect` and `disconnect` commands, it is the basic structure of all C++ programs.

```
cout << <expression> << <expression> << … << <expression>;
```
This function prints out the values of the expressions (one or more) in the console window. If any of the expressions are `endl`, the output will go to a new line at that point.

```
<type> <variable name> = <expression>;
```
This is how C++ defines a variable of the specified type (e.g., `double`, `int`, `string`). The value generated by `<expression>` will become the initial value of `<variable name>`.

```
<variable name> = <expression>;
```
This is how C++ assigns values to variables. The value generated by <expression> will become the new value of <variable name>.

```
<expression> <= <expression>
```
This condition is true if the first expression is less than or equal to the second. Other relations include `>=` (greater or equal), `<`, `>`, `==` (equal) and `!=` (not equal).

```
<variable>++
```
Increments <variable>, that is, increases the value of <variable> by 1.

Repetition

```
for (<initialization>; <continuation>; <incrementation>) {
   <do something>
   <do something>
   ...
}

while (timeRemaining(<seconds>)) {
    <do something>
    <do something>
    ...
}
```

```
while (true) {
   <do something>
   <do something>
   ...
}
```

These are different ways of doing repetition in C++. The first version will initialize a variable and repeat the loop body, executing the `<incrementation>` at the end of each repetition, and continue repeating so long as the continuation condition remains true. The second version will carry out the body for `<seconds>` amount of time. `timeRemaining` is a Myro function (see above). The last version specifies an un-ending repetition.

Chapter 4

`true, false`
These are Boolean or logical values in C++. C++ also defines true as 1 and false as 0 and they can be used interchangeably.

`<, <=, >, >=, ==, !=`
These are relational operations in C++. They can be used to compare values. See text for details on these operations.

`&&, ||, !`
These are logical operations. They can be used to combine any expression that yields Boolean values.

`%`
The modulus operator. The expression `<dividend> % <divisor>` returns the remainder after dividing integer `<dividend>` by integer `<divisor>`.

`rand()`
Returns a random integer between 0 and `RAND_MAX` (inclusive). This function is a part of the `cstdlib` library in C++.

`RAND_MAX`
A constant, which is the maximum random integer that can be returned by the `rand()` function in C++.

Chapter 5

```
if (<CONDITION>) {
   <statement-1>
  ...
   <statement-N>

}
```
If the condition evaluates to `true`, all the statements are performed. Otherwise, all the statements are skipped.

```
return <expression>;
```
Can be used inside any function to return the result of the function.

```
<TYPE> <NAME> [] = { <VALUES> };
```
Declares a C++ array called `<NAME>` with elements of type `<TYPE>`. The array is initialized to the specified `<VALUES>`.

```
char <NAME>;
```
Declares character variable `<NAME>` (able to hold one character).

Vectors and Lists:

```
#include <vector>
#include <list>
```
Includes definitions for C++ vectors or lists, which are both examples sequence containers.

```
vector< <TYPE> > <NAME>;
```
Defines an initially empty vector called `<NAME>` with elements of type `<TYPE>`.

`list< <TYPE> > <NAME>;`
Defines an initially empty list called `<NAME>` with elements of type `<TYPE>`.

```
vector< <TYPE> > <NAME> (<ARRAY START>, <ARRAY END>);
list< <TYPE> > <NAME> (<ARRAY START>, <ARRAY END>);
```
Defines a vector or list called `<NAME>` with elements of type `<TYPE>` initialized with values from location `<ARRAY START>` up to, but not including, `<ARRAY END>`. These may be of the form `<ARRAY NAME> + <CONSTANT>`.

`<seq>.size()`
Returns the current size (length) of the vector, list, or string `<seq>`.

```
<vector>[i]
<string>[i]
```
Returns the `i`th element in the `<vector>` or `<string>`. Indexing starts from 0. (Strings are a lot like vectors of characters.)

`<seq>.push_back(<value>);`
Appends the `<value>` at the back (end) of vector, list, or string `<seq>`.

`<list>.push_front(<value>);`
Appends the `<value>` at the front (beginning) of `<list>`.

`<list>.sort();`
Sorts the `<list>` in ascending order.

`<list>.reverse();`
Reverses the elements in the list.

`<seq>.begin()`
Returns an iterator referring to the first element of the vector, list, or string `<seq>`.

`<seq>.end()`
Returns an iterator referring the last element of the vector, list, or string `<seq>`.

```
vector< <TYPE> >::const_iterator <NAME>;
vector< <TYPE> >::iterator <NAME>;
list< <TYPE> >::const_iterator <NAME>;
list< <TYPE> >::iterator <NAME>;
string::const_iterator <NAME>;
string::iterator <NAME>;
```

Declares an iterator called `<NAME>`, which can refer to locations in a vector/list with elements of type `<TYPE>` or in a string. A `const` (constant) iterator does not allow the elements of the vector or list to be modified, whereas a non-`const` iterator does.

```
<iterator>++
```

Increments `<iterator>` to refer to the next element of a vector, list, or string.

```
* <iterator>
```

Returns the value stored in the vector, list, or string element referred to by `<iterator>`.

```
<seq>.insert( <it1>, <it2-begin>, <t2-end> );
```

Inserts elements from one vector, list, or string into the vector, list, or string `<seq>`. The new elements are inserted at a location specified by iterator `<it1>`. For example, use `<seq>.begin()` to insert at the beginning of `<seq>` or use `<seq>.end()` to insert at its end. The elements to be inserted are specified by iterators `<it2-begin>` and `<it2-end>`. For example, to insert all the elements from vector, list, or string `<seq2>`, use `<seq2>.begin()` and `<seq2>.end()`.

```
for ( <it> = <seq>.begin(); <it> != <seq>.end(); <it>++) {
  … STATEMENTS using <it> or *<it> …
}
```

Executes the loop body with iterator `<it>` referring to the elements of vector or list `<seq>` from its beginning up to its end.

Streams:

```
#include <fstream>
```
Includes definitions for C++ streams for file input/output.

```
ifstream <name> (<string>);
```
Declares <name> to be an input file stream connected to file name <string>.

```
ofstream <name> (<string>);
```
Declares <name> to be an output file stream connected to file name <string>.

```
<ifstream> >> <var>;
```
Reads input value from stream <ifstream> and stores it in variable <var>.

```
<ofstream> << <expression>
```
Outputs values of <expression> to output stream <ofstream>.

```
<stream>.close();
```
Closes (input or output) file stream <stream>, thus completing operation on it. You should close files when you are done with them.

```
<ifstream>.get (<chvar>);
```
Puts the next character from input stream <ifstream> into character variable `<chvar>`.

```
<ifstream>.eof()
```
Returns true if input stream <ifstream> is at the end of the file, and false if not.

Chapter 6

The `if`-statement in C++ has the following forms:

```
if (<condition>) {
   <this>
}


if (<condition>) {
   <this>
} else {
   <that>
}


if (<condition-1>) {
   <this>
} else if (<condition-2>) {
   <that>
} else if (<condition-3>) {
   <something else>
...
...
} else {
   <other>
}
```

The conditions can be any expression that results in a `true`, `false`, `1`, or `0` value. Review Chapter 4 for details on writing conditional expressions.

Chapter 7

```
struct <structure name> { <declaration>; <declaration>; … };
```

Defines a structure called `<structure name>` with *members* declared by the `<declarations>`s (usually variable and function declarations).

```
<structure name> <variable name>, <variable name>, …;
```

Defines variables belonging to a structured type. Each variable has the members declared in the preceding `<structure name>` definition.

```
<structure>.<variable name>
<structure>.<function name> ( <arguments> )
```

Accesses named member-variable (field) or member-function of a `<structure>`, which must have a structured type that declares the referenced member variable or function.

```
printf (<format string>, <expression>, … );
```

Prints the `<expression>`s according to the `<format string>`. Within it, `%W.Df` prints a floating-point number in a field of minimum width `W` with `D` digits after the decimal point, `%Wd` prints an integer in a field of minimum width `W`, and `%Ws` prints a string in a field of minimum width `W`. The value is right-justified in the field, unless a minus sign follows the `%` (e.g., `%-W.Df`, `%-Wd`, `%-Ws`), in which case it's left-justified. There are many other format specifications besides these. In a `<format string>`, `%%` prints a single percent sign.

The math library module provides several useful mathematics functions. Some of the commonly used functions are listed below:

`ceil(x)` Returns the ceiling of x as a float, the smallest integer value greater than or equal to x.

`floor(x)` Returns the floor of x as a float, the largest integer value less than or equal to x.

`exp(x)` Returns e^x.

`log(x[, base])` Returns the logarithm of x to the given base. If the base is not specified, return the natural logarithm of x (i.e., $\log_e x$).

`log10(x)` Returns the base-10 logarithm of x (i.e. $\log_{10} x$).

`pow(x, y)` Returns x^y.

`sqrt(x)` Returns the square root of x ($\sqrt{x}$).

Trigonometric functions

`acos(x)` Returns the arc cosine of x, in radians.

`asin(x)` Returns the arc sine of x, in radians.

`atan(x)` Returns the arc tangent of x, in radians.

`cos(x)` Returns the cosine of x radians.

`sin(x)` Returns the sine of x radians.

`tan(x)` Returns the tangent of x radians.

`cosh(x)` Returns the hyperbolic cosine of x.

`sinh(x)` Returns the hyperbolic sine of x.

`tanh(x)` Returns the hyperbolic tangent of x.

Chapter 8

In this chapter we presented informal *scope rules* for names in C++ programs. While these can get fairly complicated, for our purposes you need to know the distinction between a *local name* that is local within the scope of a function versus a *global name* defined outside of the function. The text ordering defines what is accessible.

Chapter 9

There were no new C++ features introduced in this chapter.

Chapter 10

```
TYPE & NAME
```
Indicates that the specified parameter is to be passed by reference (rather than passed by value), which allows it to be modified from within the function.

```
typedef TYPE NAME
```
Allows `NAME` to be used as an abbreviation for `TYPE`.

Chapter 11

```
exit(EXIT_MODE);
```
May be invoked anywhere to exit the program immediately. If `EXIT_MODE` is `EXIT_SUCCESS` then the program exits normally (equivalent to invoking `return(0)` from `main()`); if it is `EXIT_FAILURE` then the program signals abnormal termination to the operating system.

```
#include "FILENAME.h"
```
Inserts the contents of `FILENAME.h` at the above point in your code, which is a simple way of including function and other definitions in your program.