Some Basic C++ For C# and Java Programmers:

This is mostly taken from <http://www.cplusplus.com/doc/tutorial/> which has a more comprehensive reference but I’ve trimmed it down a bit. Feel free to use more if you are so inclined.

# Files

Files are typically named .cpp and .h (.h for headers, .cpp for the code that implements the headers).

# Variables and Types

Pretty much the same as C# and Java,

int a = 5;

int b = 2;

a = a + 1;

int result = a - b;

Valid variable names again, pretty much like every other language, letters, digits, underscores, no spaces. There are some reserved keywords, but those are mostly self-explanatory.

Modern versions of C++ support two interesting types, auto and decltype these two are slightly interesting. Auto tries to infer a type from some other value, decltype copies the type of some other variable. E.g.

int foo = 0;

auto bar = foo; // same as int bar = foo

decltype(foo) cat; // same as int cat;

## Casting

For those of you not used to C/C++/Java, converting between types (particularly numerical types) needs to be done explicitly, and there are a bunch of rules. I’m not going to bother with the rules, most will make sense or are easy enough to figure out, but they exist. The most obvious is converting int to/from floats/doubles.

int i;

float f = 3.14;

i = (int)f;

// OR

i = int(f);

# Basic Input/output

You’ve seen these above, for text/numerical output (either to the console or to/from a file) there are two commands cin and cout which require using <iostream> and using namespace std; (both of those are before ‘main’.

## Standard Output

cout << "Hello"; // prints Hello

cout << Hello; // prints the content of variable Hello

All of that outputs to one line until you tell it there’s a new line. E.g.

cout << "First sentence.\n";

cout << "Second sentence.\nThird sentence.";

outputs:

First sentence.  
Second sentence.  
Third sentence.

You can also end lines with “endl”

cout << "First sentence." << endl;

cout << "Second sentence." << endl;

Prints:

First sentence.  
Second sentence.

## Standard input:

int age;

cin >> age;

Input is not actually sent under the user presses “enter”

You can also chain inputs together, this is handy for reading from files e.g.

cin >> a >> b

is equivalent to

cin >> a;

cin >> b;

# Program structure

C++ is really not much different than Java or C# or python in in terms of program structure.

## If statements

if (condition) statement

if (x == 100)

cout << "x is 100";

You can combine multiple conditions with operators such as && and ||

You can nest if statements and you can also have if – else if statements

if (x > 0)

cout << "x is positive";

else if (x < 0)

cout << "x is negative";

else

cout << "x is 0";

## Switch

|  |  |
| --- | --- |
| Switch statement | Equivalent If statement |
| switch (x) {  case 1:  cout << "x is 1";  break;  case 2:  cout << "x is 2";  break;  default:  cout << "value of x unknown";  } | if (x == 1) {  cout << "x is 1";  }  else if (x == 2) {  cout << "x is 2";  }  else {  cout << "value of x unknown";  } |

## For loops

These days C++ has caught back up to C# and Java and for loops work largely the same way

The classic for loop is something like this:

for( int a = 10; a < 20; a = a + 1 ) {

cout << "value of a: " << a << endl;

}

Which prints (each on a new line, Value of a: 10, Value of a: 11, … up to 20)

C++ now supports essentially a foreach loop, though not the foreach keyword like in C#/Java

int arr[] = { 10, 20, 30, 40 };

// Printing elements of an array using

// foreach loop

for (int x : arr)

cout << x << endl;

The same logic works in C# and Java, just the cout/System.out.Print/Console.Writeline are different.

## While loops

while (n>0)

Or the like, otherwise the same as for loops

Functions (basically methods in C# and Java)

Strictly speaking *functions* and *methods* are supposed to be different things – *free functions* are not members of a class, *member functions* of a class are just called functions in C++, but they are called methods in C# and Java.

#include <iostream>

using namespace std;

int addition (int a, int b)

{

int r;

r=a+b;

return r;

}

int main ()

{

int z;

z = addition (5,3);

cout << "The result is " << z;

}

Prints: the result is 8.

As in other languages, functions can be called multiple times, functions can call other functions etc. Pretty standard stuff.

# Compound Data Types

## Arrays

Arrays in C++ are pretty much the same as in C#/Java

int foo [5];

creates an array with 5 elements (starting at foo[0]) and

int foo [5] = { 16, 2, 77, 40, 12071 };

Creates the array and gives it 5 values.

#include <iostream>

using namespace std;

int foo [] = {16, 2, 77, 40, 12071};

int n, result=0;

int main ()

{

for ( n=0 ; n<5 ; ++n )

{

result += foo[n];

}

cout << result;

return 0;

}

Creates an array and prints the sum of the array.

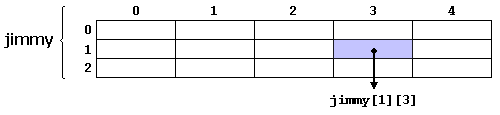
## Multidimensional Arrays

Again, same as C#/Java largely:

int jimmy [3][5];

is a 3 row 5 column array

jimmy[1][3]



## Passing Arrays

void procedure (int arg[])

int myarray [40];

procedure (myarray);

Passes the address of myarray to the procedure, as ‘arg’.

It’s common to pass an array and its length together

#include <iostream>

using namespace std;

void printarray (int arg[], int length) {

for (int n=0; n<length; ++n)

cout << arg[n] << ' ';

cout << '\n';

}

int main ()

{

int firstarray[] = {5, 10, 15};

int secondarray[] = {2, 4, 6, 8, 10};

printarray (firstarray,3);

printarray (secondarray,5);

}

For example

## \*\*\*\*\*\*Pointers and References\*\*\*\*\*

*Pointers* and *references* are not something you see in C#, Java, Python or R. They exist in C# and references are in Java but the compiler generally takes care of the reason for pointers and references in the first place, so you don’t tend to see people using them.

## Address-of Operator (&)

Address the memory address of something to a value – this means you can pass around the memory address of a large object in memory, rather than making a copy of it or moving it.

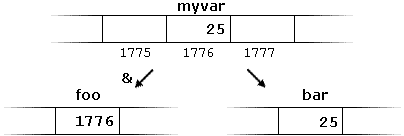
foo = &myvar;

Consider:

myvar = 25;

foo = &myvar;

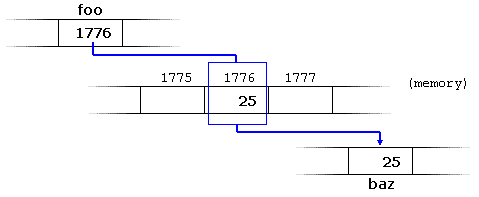
bar = myvar;



## Dereference Operator (\*)

A variable which stores an address of an object is called a pointer, to actually get at the object pointed to, you need to dereference the pointer:

baz = \*foo;



baz = foo; // baz equal to foo (1776)

baz = \*foo; // baz equal to value pointed to by foo (25)

## Pointers and arrays

Pointers and arrays are pretty interrelated. Really the variable for an array points to the start of the array, and the math of how that works is hidden from you.

int myarray[20];

int\* mypointer;  
  
creates an array, and a pointer

mypointer = myarray;

makes the two essentially equivalent, except that mypointer could be changed to point to a different array, while myarray cannot have that happen.

int numbers[5];

int\* p;

p = numbers; \*p = 10;

p++; \*p = 20;

p = &numbers[2]; \*p = 30;

p = numbers + 3; \*p = 40;

p = numbers; \*(p + 4) = 50;

for (int n = 0; n < 5; n++)

cout << numbers[n] << ", ";

Will print 10, 20, 30, 40, 50.

## Function pointers

A feature we probably aren’t (and shouldn’t be) using is function pointers. These are taken from functional languages and let you do some really cool powerful things which are mostly irrelevant to HPC.

But they are super cool.

A function pointer is a pointer to a function – meaning you can pass a function (well a pointer to a function) to a method, transform or execute the function, and then return a result.

Consider the following code:

#include <iostream>

int foo() // code starts at memory address 0x002717f0

{ return 5; }

int main()

{

std::cout << foo; // we meant to call foo(), but instead we're printing foo itself!

return 0;

}

In C++ that prints… whatever the address of Foo is, not 5. For Sri that was 002F1424 but it changes each time I run the program (the place I stole it from has the value in comments)

Ah, but that address is the pointer to the code.

So there’s a bunch of neat stuff you can do here, I’ll skip to the good stuff but for thorough reading:  
<https://www.learncpp.com/cpp-tutorial/78-function-pointers/>

You can do things like have user defined functions (or at least user selected).

#include <algorithm> // for std::swap, use <utility> instead if C++11

#include <iostream>

// Note our user-defined comparison is the third parameter

void selectionSort(int\* array, int size, bool (\*comparisonFcn)(int, int))

{

// Step through each element of the array

for (int startIndex = 0; startIndex < size; ++startIndex)

{

// bestIndex is the index of the smallest/largest element we've encountered so far.

int bestIndex = startIndex;

// Look for smallest/largest element remaining in the array (starting at startIndex+1)

for (int currentIndex = startIndex + 1; currentIndex < size; ++currentIndex)

{

// If the current element is smaller/larger than our previously found smallest

if (comparisonFcn(array[bestIndex], array[currentIndex])) // COMPARISON DONE HERE

// This is the new smallest/largest number for this iteration

bestIndex = currentIndex;

}

// Swap our start element with our smallest/largest element

std::swap(array[startIndex], array[bestIndex]);

}

}

// Here is a comparison function that sorts in ascending order

// (Note: it's exactly the same as the previous ascending() function)

bool ascending(int x, int y)

{

return x > y; // swap if the first element is greater than the second

}

// Here is a comparison function that sorts in descending order

bool descending(int x, int y)

{

return x < y; // swap if the second element is greater than the first

}

// This function prints out the values in the array

void printArray(int\* array, int size)

{

for (int index = 0; index < size; ++index)

std::cout << array[index] << " ";

std::cout << '\n';

}

int main()

{

int array[9] = { 3, 7, 9, 5, 6, 1, 8, 2, 4 };

// Sort the array in descending order using the descending() function

selectionSort(array, 9, descending);

printArray(array, 9);

// Sort the array in ascending order using the ascending() function

selectionSort(array, 9, ascending);

printArray(array, 9);

return 0;

}

Allows the user to pick which sort function is used (ascending or descending, the demo just uses both).

## \*\*\*\*\*Dynamic Memory\*\*\*\*\*

In C#, Java and C++ there is a keyword “new” (in C this accomplished with the more complicated ‘malloc’) that creates an object in dynamic memory space, by allocating memory for it, and initialising it.

A key difference with C#/Java and C++ is that when you’re done with an object in C++ you must explicitly delete it, whereas in other languages that is taken care of for you.

int \* foo;

foo = new int [5];

The previous way we declared arrays forced arrays to a fixed size at compile time. This allows arrays to have their size chosen at run time (by passing a variable).

This declaration can fail, and there’s some error checking for it, but hopefully we won’t encounter that problem.

## \*\*\*\*\*Delete\*\*\*\*

When you are done with an object in C++ (if you aren’t just finishing the program) you need to free up that space by deleting all of the objects. This is where memory leaks happen and a big part of learning to program C++ for complicated programs well (ironically HPC doesn’t necessarily have this constraint since it’s largely about a handful of numerical methods at a time).

The basic syntax:

delete pointer;

delete[] pointer;

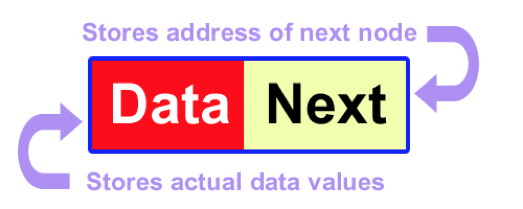
The first deletes a single object, the second deletes an array of objects.

If you need more advanced data structures you will need to make sure you correctly delete their contents when being done with the object.

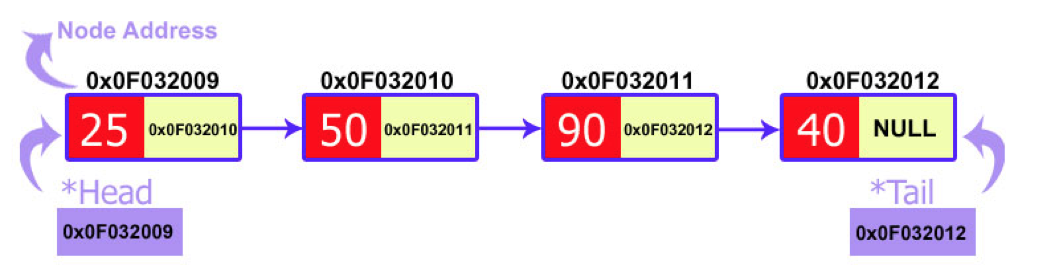
## Data structure: Linked List in C++

A linked list is a collection of nodes with pointers to other nodes (a singly linked list is a node and a pointer to the next node in the least). This also means you need a pointer to the first node in the list.

From <https://www.codementor.io/codementorteam/a-comprehensive-guide-to-implementation-of-singly-linked-list-using-c_plus_plus-ondlm5azr>



That’s usually a struct (or a class) node the list itself is like this:



#include <iostream>

struct node

{

int data;

node\* next;

};

class LinkedList

{

private:

node\* head, \* tail;

public:

LinkedList()

{

head = NULL;

tail = NULL;

}

void display()

{

node\* temp = new node;

temp = head;

while (temp != NULL)

{

std::cout << temp->data << "\t";

temp = temp->next;

}

}

void insert\_start(int value)

{

node\* temp = new node;

temp->data = value;

temp->next = head;

head = temp;

}

};

int main()

{

LinkedList\* DemoList;

DemoList = new LinkedList();

for (int i = 0; i < 10; i++)

DemoList->insert\_start(i);

DemoList->display();

//std::cout << "Hello World!\n";

return 0;

}

So now you could build a linked list (or a doubly linked list). For more details see COIS 2020H on different kinds of data structures etc.