# Strings

The compiler continues to support null terminated strings.

char\*p = “Hello”; // 6 bytes … letters plus a ZERO on the end to mark the end

C string library to work with these.

Have to keep up with the zero on the end and remember when to delete the pointer.

Common errors: overflowing by one byte because of the terminator.

If there is time, do an exercise where you make a string class.

void main() {

string one = "Hello World"; // Constructor that takes a char\*

string two = one; // Show where things live

one[0] = 'P'; // Operator overload to change characters

cout << one << endl;

cout << two << endl; // Two "P" didn't change. This is a copy.

int len = one.size(); // Get the length

// Operator+ overload for appending and assignement

// Space is created ... old space deleted.

one = one + two + two;

if(one == two) {

// == is overloaded to do the string compare

}

int i = one.find('o'); // First 'o'

int j = one.find('o',5); // First 'o' starting with one[5]

string sub = one.substr(6,3); // Starting index and count ... not end index

system("pause");

}

void main() {

string arr[4]; // Create FOUR objects using their default constructor

arr[1] = "One";

arr[3] = "Last";

for(int x=0;x<4;++x) {

cout << ":" << arr[x] << ":" << endl;

}

system("pause");

}

void main() {

// Some string processing

string text = "I am a %replaceMe% test.";

string rep = "rep";

int i = text.find('%');

int j = text.find('%',i+1);

cout << i << " " << j << endl;

int repLength = j-i+1; // Why plus one?

string firstPart = text.substr(0,i);

string secondPart = text.substr(j+1);

text = firstPart + rep + secondPart;

cout << text << endl;

system("pause");

}

include <iostream>

#include <string>

#include <sstream>

using namespace std;

//void main(int argumentCount, char\*\* arguments) { // usually argc, argv

void main(int argumentCount, char\* arguments[]) {

for(int x=0;x<argumentCount;++x) {

cout << arguments[x] << endl;

}

// First is the name of the program

int a = atoi(arguments[1]); // For C strings

string one = "1234";

int b = atoi(one.c\_str());

// You already know a text to int converter .. the ">>" with int.

// Need a way to make a stream out of a string.

stringstream ss(one);

int i;

ss >> i;

// From <string> ... read a line from an istream into a string object. Don't have to mess

// with size since the string can grow and grow.

getline(cin,one);

}