Cs111 8/30/2017

* Program 1

1. Goal: say hello to a user/ to the screen

C++ of user’s screen is called **cout** (c-out) c++ programs are lists of instructions carried out in order

1. Cout: **cout** (where output goes) **<<** (send something after this operator) **;** (c++ end a sentence with a ;)
2. Standard beginning: the hello world program

**#include <iostream>**

using namespase std;

int main() {

cout << “Hello word!” << endl;

return 0;

}

Week 1 lab goals:

* Files (used to store information like our program) and directories on venus
* An editor to make and edit files Editors: vi, emacs, pico - to enter data to files
* SSH allows you to connect to venus from anywhere/ SFTP
* g++ compiler

1. Venus

9/6/2017

Every c++ program has a standard start and std end ( #include < >, return )

In between is a list of instructions to perform in order.

* instruction of output:

1)Anything inside “human words ” is copied as written without trying to understand it.

2)endl means make a new line

Goal: 1.Make c++ read what we type on screen

2. Make c++ store data in its memory

A. A variable: is a small box in c++’s memory hat can store data

Variable might have different sizes. For every piece of data we have to give c++ a type so it can set a sensible box.

CS111 only 5 type variable (5 of 64 words of c++):

1. int: stores whole numbers (e.g: 2,3,4)
2. double: decimal numbers (e.g 3.1415926)
3. char: single characters (A & $)
4. bool: true false values
5. string: text

Before use a variable (to store data) we must decide (and tell c++) its type and its name ----- **need a declaration for c++**

type name; (e.g int number;)

(Also allowed: short, long, float)

B.To get data from screen (cin):

cin >> variable to store what user type;

Goal: venus > ./a.out (to make the user type data we should always print a prompt)

9/11/2017

Variables, declare before using [type name;]

Primitive types: int double bool char string

Name rules: a name is made of letters and digits. Must begin with a letter \***begin with a number is not valid**

**C++ matters the case of letters (Advice:** ONLY use lower case letters in C++**)**

this guarantees correct names and gives readers a clue that its C++

Topic: Arithmetic

A. In c++ + means add; - means subtract; \* means multiply

**1.\*Work with usual algebra rules e.g:** cout << 1+2\*3; ---- 7

Can put in () to override normal precedence e.g: cout << (1+2)\*3;

**2.Division in c++ written /** (divide has 2 meanings for us but not for c++)

a) Int division because answer is an int ----- 2 into 5 goes (2 times with a remainder of 1)

Decimal division ----- 5/2=2.5

Tell c++ to divide two ints it does int division

b) If one or both things are decimal it dose decimal division

Cout << 5/2.0;

Cout << 5.0/2;

Cout << 5.0/2.0;

All above get an answer of 2.5

2 and 2.0 are different in c++ types (must be careful about types!)

1. C++ rule: the result of + - \* or / is an int if both parts are int otherwise it’s double.

E.g: c = (f-32)\*5/9

//convert temperature to Celsius

#include <iostream>

Using namspace std;

Int main() {

Int f;

Double c;

Cout << “What is the temperature in F: ”;

Cin >> f;

C= (f - 32) \* 5/9.0; //= is called assignment [VAR]=[Expression] c++ calculate the expressions and puts answer into the variable

Cout << “In Celsius that is ” << c << endl;

Return 0;

}

9/13/2017

= is to change a value of a variable

/ has two meaning [1. int division 2. exact division]

**Int division --- 7/2 [e.g: 2 goes into 7, 3times with reminder 1]**

Get a reminder ---- 7%2 [% is a new symbol to get a reminder. Pronounce % as “modulo”]

1. Arithmetic uses =, +, -, \*, /, %

P.94 e.g:

#include <iostream>

Using namespace std;

Int main() {

Int number, q, d, n, c; //q=quarter d=dime n=nickel c=pennies

Cout << “How many cents in change?”;

Cin >> number;

q = number/25;

number = number % 25;

D = number/10;

{in code file in F named as the date}

Rule: In c++ the computer ignores spaces, tabs and newlines (except to separate words)

Chapter 2 Decisions

Model: if ([yes/no question]) [special action];

E.g: if (age > 100) {

Cout << “liar!” << endl;

(\n = newline \t = tab can put inside the “” [\\backslash)](\\\\backslash))

**2 way decision**

If ([y/n Q]) [action];

Else [action2];

9/18/2017

Simple decision: if statement

Determine whether a # is odd or even

If( (n%2) >0 )

cout << “odd.\n”;

Else

cout << “Even.\n”;

A. a > b test whether a is greater than b

a < b test whether a is less than b

a >= b test whether a is greater or equal to b

a <= b test whether a is less or equal to b

a != b test whether a is different from (or not equal to) b

a == b is a equal to b

Goal: Ask user to enter even # if they mess up make them try enter again

If ((n%2) != 0)

Cout << “Try again: ”;

Cin >> n;

In c++ we can group multiple actions as a single “compound action” or **block by enclosing them in { }**

B. We need to be able to **repeat questions** until we get the answer we want

**while statement:** while( [y/n question] ) action; (repeat question)

If statement (single question)

E.g: Ask for odd # ask repeatedly until we get odd #

Int main () {

Int n;

Cout << “Give me an odd # ”;

Cin >> n;

While ((n%2)==0){

Cout << “Idiot I said odd: ”;

Cin >> n;

}

Cout << “Thank you.\n”;

Return 0;

}

Goal: ask user for even # give them 3 goes to be right then kick them out

int main(){

Int n, strikes = 0; (**initialize**: when you declare a variable you can supply a starting value)

Cout << “Even # please: ”;

While ((n%2) !=0){

Strikes = strikes + 1;

If (strikes == 3){

Cout << “you are out!\n”;

}Return 0;

Else {

Cout << “That is odd. Try again: ”;

Cin >> n;

}

}

Cout << “Thank you! \n”;

Return 0;

}

9/25/2017

Repeat decision or action: while sentence\*9/18

E.g say “hello” 10 times

Int main(){

Int counter = 1;

While(counter <= 10){

Cout << “Hello \n”;

Counter = counter + 1;

}

Return 0;

}

1. Goal: print #1,2,3...

Int main(){

Int c = 0;

While(c <= 100){

Cout << c << endl;

C = c +1

}

Return 0;

}

1. Goal: show powers of 2 up to 20

Int main(){

Int power = 2, n = 1;

While(n <= 20){

Cout << power << endl;

Power = power \* 2;

N = n +1;

}

Return 0;

}

1. Goal: user gives # we print backwards
2. Get the users number n 2.print n % 10

Int main() {

Int n;

Cout << “Enter a number: ”;

Cin >> n;

While(n > 0){

Cout << n%10;

N = n/10;

}

Return 0;

}

1. Goal: user enter # we say if it is prime

Not prime = has smaller factor 15=3x5

Prime = don’t have small factor

Int main(){

Int n, factor = 2;

Cout << “enter a number: ”;

Cin >> n;

While(factor < n){

If(n%factor == 0){

Cout << “not prime.\n”;

Return 0;

}

Factor = factor + 1;

}

Cout << “Prime”;

Return 0;

}

9/27/2017

We can make more complex questions by using “and” “or” “not”

&& || !

If((age > 100) || (age < 0)) if(!(age > 18)) if((age>= 13) && (age<= 19)

Cout << “lair\n”; cout << “you can use this”;

Chapter 4:

**for loops: another way to repeat actions**

C++ lets us do this with a for loop

Model: for([initial Var]; [valid condition]; [change condition]) has exactly same effect as the while loop

action;

1. Goal: squares #

Int main() {

For(int c = 1; c <= 100; c++)

Cout << c\*c << endl;

Return 0;

}

1. goal: user enters c we print 25th power

Int main() {

Double x;

Cout << “enter decimal #: ”;

Cin >> x;

Double ans = 1.0;

For(int c = 1; c <=25; c++)

Ans\* = x; // ans = ans\*x

Cout << ans << endl;

Return 0

}

1. goal: line of 50 \* s

Int main() {

For(int c = 1; c <= 50; c++){

Cout << “\*” << endl;

}

Return 0;

}

4. goal: find sum of odd#

Int main(){

Int sum = 0;

For(int c = 1; c <= g; c += 2)

Sum += c;

Cout << sum << endl;

Return 0;

}

1. sum of first 100 odd numbers

Int main(){

Int sum = 0, odd = 1;

For(int c = 1; c <= 100; c++ ){

Sum += odd;

Odd += 2;

}

Cout << sum << endl;

Return 0;

}

10/02/2017

Model:

for([initialize value]; [condition valid]; [change condition] ){

[action]

}

Example 1: print 10x10 square of \*

Int main(){

For(int r = 1; r <10; r++){

For(int c = 1; c <= 10; c++){

Cout << “\*”;

Cout << endl;

}

}

Return 0;

}

A nested loop uses an “inner loop” as part of the action of an “outer loop”. A for loop gives your power to make c++ repeat.

Example 2: \*Space rule :**After you met a { indent all statement by 3 extra spaces until you reach the closing }**

Goal: triangle of \*

Int main(){

For(int r = 1; r <= 10; r++ ){

//make row r with r star & endl

For(int c = 1; c <= r; c++){

Cout << “ \* ”;

Cout << endl;

}

}

Return 0;

}

Example 3: upside down triangle

Int main(){

For(int r = 10; r <= 1; r- -){

}

}

10/04/2017

Midterm 1: 10/18/2017 9am last name A-J Sb c201 K-Z here

example 1:

for(int r = 1; r <= 5; r++){//r=roll

for(int c = 1; c <= 5; c++){

if((c == r)|| ((r + c) == 6)) cout << "\*";

else cout << " ";

}

cout << endl;

}

Example 2:

for(int r = 1; r <= 5; r++){

for(int c = 1; r <= 3; c++){

if((r == c) || ((r + c)== 6)) cout << "\*";

else cout << " ";

}

cout << endl;

}

Example 3:

for(int r = 1; r <= 20; r++){

//make row r

//it needs r terms like cube c

//an = sign and sum

int sum = 0;

for(int c = 1; c <= r; c++){

cout << "cube " << c;

sum += c\*c\*c;

if(c < r) cout << " + ";

else cout << " = ";

}

cout << sum << endl;

}

2 ways to loop

for and while (**If you know in advance how many repeats --- for otherwise ---- while**)

Example 4

Print all #s up to 1000 whose digits sum to 13

10/11/2017

Chapter 5: Functions

A function is a small piece of code to perform a single useful task that can be called on from our programs when the task is needed.

Math functions and many other start with input and calculate an output.

Random function call on rand and it makes up a random # for us (no input but there is output)

How do we make our program call on functions? How do we make function to call on?

To able to call on a function we need to know, 3 principal attributes:

1. it’s name (e.g sqrt( ) is square root || rand( ) is random number function)
2. What input is required(if any)
3. What answer will be found (if any)

\*We can call on a function from anywhere in our code

[function name] (supply input)-----c++ performs the task and replace the call by its result

Example 1.

#include <iostream>

#include <cmath> //collection of math function

using namespace std;

int main( ){

For(int c = 1; c <= 10; c++)

cout << c << “ ” << sqrt (c) << endl;

return 0;

}

Example 2.

Roll 5 dice output 1 5 2 1 6

[start] rand( ) %[range] produce a range of #s beginning at start

#include<iostream>

#include<cstdlib>

Using namespace std;

Int main( ){

For(int n = 1; n <= 5; n++)

Cout << 1 + rand( )% << endl;

Return 0;

}

In c++ random #s start the same way every time we run a program “pseudo random” c++ allows us to start the function srand( ) is used to start them use once at the start of a program.

Srand( ) input: the starting random# output: void

Srand(time(0))

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

int main(){

srand(time(0));

for(int n = 1; n <= 5; n++) cout << 1 + rand() % 6 << endl;

return 0;

}

How do we write our own functions?

[title line] { //explains its attributes to c++

[code to do the task] // ordinary c++ instructions as we’ve seen all semester!!

}

Model title line:

[type of answer] [name of function] (describe input)

For sqrt:double sqrt (double x)

Rand: int rand( )

Max: int max(int x, int y)

Goal: Max function

int main(){

int a=4, b=7;

cout << max(a,b) << endl;

return 0;

}

int max(int x, int y){

if(x > y) ans = x;

else ans = y;

return ans;

}

#include <iostream>

Using namespace std;

[max function here] //place a function before it is called

[main function here]

10/23/2017

**Chapter 5 Functions**

[title line]{ model title line [name](declare parameter)

[block of code]

}

Argument: = place of input given to a function when you call on it. It is an actual piece of data, e.g a number.

max(int x, int y) input is specified by writing declarations for input values, in order.

int main(){

int myScore = 2, yourScore;

cout << "What was your score: ";

cin >> yourScore;

cout << "I got a " << max(myScore, yourScore) << endl;

return 0;

}

(x and y behave as variables they couldn’t add only # our user gives)------parameter = a piece of input as seen by the function’s writer. These are variables not particular #s.

int ans = x;

if(y > x) ans = y;

return ans; (return: stores to function ans:repeat the answers)

#include <iostream>

using namespace std; when you run a prog. C++ starts at the beginning of main

int max(int x, int y){

[copy code for max here]

int main(){

cout << max(myScore, yourScore) << endl; //at this line we call up to the function. Argument are duplicated and use

as int a variable for x,y then turn max for return and send back ans.

[copy code for main here]

10/25/2017

Before writing a function write a simple main to use it.

Example 1:

int main(){

for(int n = 1; n <= s; n++)

void sayHello( ){

cout << “Hello ” << endl;

return; //return to no answer, function type void

}

return 0;

}

Example 2:

int cube(int x){

int ans = x \* x \* x;

return ans;

}

int main(){

int s = 5;

for(int n = 1; n <= s; n++)

cout << cube( n ) << endl;

return 0;

}

The scope of a variable is the region where it makes sense. It is contained in the function where the variable lives.

void stars(int x){

for(int i = 1; i <= x; i++)

cout << “\*”;

return;

}

//triangle 5 rows

int main(){

for(int r = 1; r <= 5; r++){

stars(r);

cout << endl;

}

return 0;

}

Example 3:

int smallestFactor(int x){

for(int f = 2; f <= x; f++)

if((x % f) == 0)

return f;

return x;

}

int main(){

int n;

cout << "Enter a number greater than 1 ";

cin >> n;

if(n = smallestFactor(n)) cout << "Prime\n";

else cout << "Not prime\n";

return 0;

}

10/30/2017

When you call a function you supply input arguments which c++ calculates and turns into numbers.

C++ then switches to work on the function

Each parameter is a variable that starts with the numerical value of the argument

Eventually a return instruction ends the function and c++ returns to what it was doing

This process is known as call by value

int main(){

int x = 5;

int y = add10(x);

cout << x << " " << y << endl;

return 0;

}

int main(){

x = x +10;

return x;

}

Call by reference: A function is handed the actual, data any change it makes is permanent.

C++ can operate call by reference you request it.

Recursion: A function can be called on to help carry out any task.

-From main

-From another function

-From within itself

n!= n \* (n-1)!

factorial(int n){

if(n == 1) return 1;

return n\* factorial(n - 1);

}

11/01/2017

Recursion: A function can call on its own task

Example factorial:

int factorial(int x){

if(x == 1) return 1;

return x \* factorial(x - 1);

}

int main(){

cout << factorial(3);

return 0;

}

Rules to use recursion:

1. ONLY call on easier versions of the task
2. You MUST include a base(easiest) case and tell it to stop these
3. Try to begin by detecting and resolving the base case
4. Don’t ask how c++ can do the work

Example 2: sum of Digits

int sumDigits(int x){

if(x < 10)return x;

return x % 10 + sumDigits(x / 10);

}

int main(){

cout << sumDigits(19683);

return 0;

}

Example 3:

int triangle(int x){

if(x == 0) return 0;

triangle(x - 1);

for(int c = 1; c <= x; c++) cout << "\*";

cout << endl;

}

int main(){

triangle(5);

return 0;

}

11/06/2017

Example1:

int removeFirst(int x){

if(x < 10) return 0;

return removeFirst(x/10) \*10 + x % 10;

}

int mian(){

cout << removeFirst(19863);

return 0;

}

Remove 0s from the number:

Example 2:

void remove0s(int x){

if(x < 10) cout << x;

else{

remove0s(x/10);

if((x%10)!=0) cout << x % 10;

}

}

int main(){

remove0s(1002040);

return 0;

}

Example 3:

int removeZeros(int x){

if(x < 10) return x;

int y = removeZeros(x/10);

if((x % 10)== 0)return y;

return y \* 10 + x % 10;

}

int main(){

cout << removeZeros(1002040);

return 0;

}

11/08/2017

Chapter 6 Arrays - useful whenever we have lots of similar data items

//goal read 5#s, print in reverse

Every array has:

Name: usual name rules

Capacity # of elements it has

Base-type the type of element it contains

[base type] [name] [(capacity)];

int x[10]; get element x[0] x[1] x[2]...x[9]

Example scores:

int main(){

int scores[5], total;

double average;

string names[5] = {"Max", "Freddy", "Arthur", "Jack", "Kelly"};

for(int i = 0; i < 5; i++){

cout << "Enter quiz score for " << names[i] << ": ";

cin >> scores[i];

}

for(int i = 0; i < 5; i++) total += scores[i];

average = total / 5.0;

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

if(scores[i] < average) cout << names[i] << " is failing.\n";

}

return 0;

}

11/13/2017

Array gives many elements each is a variable it has capacity and base type.

The base type is the type of variables it gives capacity is # elements. They are numbered starting at 0 ending at capacity-1

int main(){

int data[80], cols;

cout << "Number cols at most 80? ";

cin >> cols;

cout << "Enter the data values\n";

for(int i = 0; i < cols;i++) cin >> data[i];

}

11/15/2017

2D arrays - used to store table of data

Any table have --- base type= type of data stored row capacity= # rows col capacity = # cols name

In c++ you set this up with a 2d array(which works very similarly to an ordinary or 1d array)

Declare like this {base type}(name) [row cap][col cap]

E.g. int scores[4][3];

You can access entry in row r col c as scores[r][c]

Add 1 point to everyone on quiz 2

for(int r = 0; r < 4; r++) scores[r][1] = scores[r][1]+1;

To do something with all enters we should use nested loop

Print scores:

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

for(int c = 0; c < 3; c ++) cout << scores[r][c] << " ";

}

cout << endl;

}

//class 4 students do 3 quizzes 1. read scores quiz by quiz 2. calculate average for every student 3. print average

int main(){

int scores[4][8];

double average[4];

int q, s;

for(q = 0; q < 3; q++){

for(s = 0; s < 4; s++ ){

cout << "Score for numbers ";

cin >> scores[s][q];

}

}

for(s = 0; s < 4; s++){

int total = 0;

for(int q = 0; q < 3; q++) total +=score[s][q];

average[s] = total / 3.0;

}

for(s = 0; s < 4; s++) cout << "Average for " << s << "is " << average[s] << endl;

return 0;

}

//box plot data in 1d array

int main(){

int s;

int data[s] = {3,1,4,1,5};

string picture[40][5];

int max = data[1];

for(int c = 0; c < s; c++) if(data[s] > max) max = data[c];

for(int c = 0; c < 5; c++){

for(int r = 0; r < max; r++)

if(data[c] = r) picture[r][c] = "x";

else picture[r][c] = " ";

}

for(int r = max - 1;r >= 0; r--){

for(int c = 0; c < s; c++) cout << picture[r][c];

cout << endl;

}

return 0;

}

11/20/2017

Arrays and Functions:

You can always supply(a whole) array as input for a function

1.How do we specify the array as an argument when we call the function

2.How do we specify an array parameter in the title line?

3.What else?

A)The array data is treated by reference so many change the function makes to the array data is permanent

B)The function cant detect the capacity of the array

Normally if a function uses an array as input it needs a second input to say how big the array is

Goal: functions to show first 3 elements of an array

Goal:function to return average of array

Goal:Max of array