10/30/19

break; - abandon this and all remaining iterations of the loop

continue; - abandon this iteration of the loop

void exchange (int& x, int& y);

int main ()

{

…

int a;

int b;

cin >> a >> b;

if (a < b)

exchange (a,b);

cout << a << “ is at least as big as “ << b << endl;

}

void exchange (int& x, int& y)

{

int t = x; -> cannot int& t =x -> uninitailzed

x = y;

y = t;

}

Java: passing pointer – reference

C++

Allow automatic conversion from char to int and int to char

char ch = 76; // ‘L’ if ASCII is the encoding, ‘<’ if EBCDIC

int k = ‘L’ -> not sure

k++; // k is 77

increment characters -> ch++; (next char in the encoding skin)

double d = 3.68;

cout.setf(ios::fixed);

cout.precision(1);

cout << d; // calls the function for double; writes ‘3’ ‘.’ ‘7’

// if ASCII, this is 51 46 55 or EBCDIC 243 75 247

cout << k; //print an int -> calls the function for int; writes

Print ‘7’ ‘7’

ASCII: 55 55

cout << ch; //calls the function for char; writes ‘M’ if ASCII

string s = “A7’

cout << s; call the function for string; writes ‘A’ ‘7’

// if ASCII -> 65 55

cout << s[1]; //writes ‘7’

cout << ch+1; //calls the function for int; writes ‘7’ ‘8’

//if ASCII -> ‘55’ ‘56’

NEW:

bool isValidDate (int y, int m, int d)

{

…

if (m < 1 || m > 12 || d < 1)

return false;

//february w/ leap year

if (m == 2)

{

if (y % 4 == 0)

return m <= 29;

else

return m <= 28;

}

if ((m < 8 && m % 2 ==1) || (m > 7 && (m % 2 == 0))

return d <= 31;

if (m < 8 && m % 2 == 0) || (m > 7 && (m % 2 == 1))

return d <= 30;

}

Set up a table -> daysInMonth -> 12 ints

bool isValidDate (int y, int m, int d)

{

const int daysInMonth[12] = { //constant -> NOT CHANGING

31, 28, 31, 30, 31, 30,

31, 31, 30, 31, 30, 31

}

…

if (m < 1 || m > 12 || d < 1)

return false;

//? if (y % 4 == 0 && m == 2)

return (m <= 29))

return (d <= daysInMonth[m-1]); //use pre-set functions -> complexity – simple interface is better

}

…

switch (m)

{

case 1: cout << “January”; break;

…

case 12: cout << “December”; break;

//default?

}

const string monthNames[12] = {

“January”, “Febuary”, … , “December”

}

cout << monthNames[m-1]; //assuming month 1-12 encoding skin -> m-1 to call

* New Output

const int numOfMonths = 12;

const int daysInMonth [numOfMonths] = {

31, 28, 31, 30, 31, 30,

31, 31, 30, 31, 30, 31

}

const string monthNames[numOfMonths] = {

“January”, “Febuary”, … , “December”

}

cout << “These months have 31 days:” << endl;

for (int k = 0; k < numOfMonths; k++) // or K <= numOfMonths -1 or K <= 12 – giving sym

{

if (daysInMonth[k] == 31)

cout << monthNames[k] << endl;

}

//USE CONSTANT for common values

* const int numOfMonths = 12;

Undefine behaviors if negative int in an array or over the size

… daysInMonth.size() … NO!!!

… daysInMonth.length … NO!!!

cout << “Enter the scores (negative when done): ‘’;

int total = 0;

int numOfScores = 0;

for (;;)

{

int s;

cin << s;

if (s < 0)

break;

total += s;

numOfScores++;

}

if (numOfScores == 0)

cout << “There were no scores, so no average”;

else

cout << “The average of all the scores is ‘ << 1.0\*total/numOfScores << endl;

OR  
cout << …. << static\_cast<double>(total)/numOfScores << endl;

Mean + Standard Deviation – Diff betwee mean and data -> squared

11/01/19

cout << “Enter the scores (negative when done): ‘’;

int total = 0;

int numOfScores = 0;

const int MAXSCORES = 10000; //OR 10’000

int scores [MAXSCORES];

for (;;)

{

int s;

cin << s;

if (s < 0)

break;

if (numOfScores == MAXSCORES)

{

cout << “I can handle only “ << MAXSCORES << “ scores!” << endl;

cout << “Continuing with just the first “ << MAXSCORES << “ values.” << endl;

break;

}

scores[numOfScores] = s;

total += s;

numOfScores++;

}

if (numOfScores == 0)

cout << “There were no scores, so no statistics”;

else

{

double mean = static\_cast<double> (total)/numOfScores;

cout << “The average of all the scores is ‘ << mean << endl;

double sumOfSquares = 0;

for (int k = 0; k != numOfScores; k++)

{

double diff = scores[k] – mean;

sumOfSquares += diff \* diff;

}

cout << “The standard deviation is “ << sqrt (sumOfSquares/numOfScores) << endl;

}

* Be careful of the actual capacity of the array compare to the useful values
* Check: arthemetic errors (div by 0, square root of neg) or array (non-neg or greater or equal to the capacity)
* Check loop limit -> Bug catching – proof reading codes in advance

New:

double computeMean (const int a[], int n)

void setAll (int b[], int n, int value);

double computeMean (const int a[], int n)

{

if (n <= 0)

return 0.0;

int total = 0;

for (int k = 0; k < n; k++)  
{

total += a[k];

}

return static\_cast <double> (total) / n

}

int main ()

{

…

const int MAXSCORES = 10000;

int scores [MAXSCORES}

int numOfScores = 0;

…//fill arrays

double m = computeMean (scores, numOfScores)

…

int stuff [100];

…

double m2 = computeMean (stuff, 10000)

…

}

Array – never pass by values (not pass by copying) -> location

Use passing a number greater than the array size

* NOTHING you can do if the user enters a bigger number than the array

DO NOT USE SIZE OF – wrong result – unable to find how large the size of array is

* Assignment of array

void setAll (int b[], int n, int value)

{

for (int k = 0, k < n, k++)

{

int b[k] = value;

}

ex. computeMean(…) -> able to compute

how. CANNOT call setAll in computeMean – will not compile - WRONG

}

Ex. setAll(stuff, 50, -42)

const int daysInMonth[12] = {31, 28, 31… 31};

cout << computeMean(daysInMonth, 12) << endl;

However:

cout << setAll(daysInMonth, 12, 30) << endl; //UNABLE TO COMPLIE – cannot modify

enforce by complier for const -> easy to see

----------------------------------------------------------------------------------------------------------------

STRING in C++

#include <iostream>

#include <string>

using namespace std;

string t = “Hi”;

string s; //empty string (string s = “”)

for (int k = 0; k != t.size(); k++)

cout << t[k] << endl;

cout << t;

getline(cin, s);

s = t; //assign strings – set s to “Hi”

s += “!!!” //add !!! to s -> s = “Hi!!!”

Compare strings:

if (t < s)

// t = “Hello” // s = “Hello!!!”

If length ends at the same time with exact same character for all -> equal string

Keeps going: the one that runs out first will be less than the ones that keeps on going

* s is greater than t, so (s >t == true)

Different characters

* lowercase l is less than lowercase p
  + First difference – compare Hello to Help -> l is less

‘0’ < ‘1’ < ‘2’ … ‘8’ < ‘9’ -> contiguous values – no other values in between

DO NOT guarantee contagious values (may have characters in between)

‘a’ < ‘b’ < ‘c’ < ‘d’ … ‘y’ < ‘z’

‘A’ < ‘B’ < ‘C’ < ‘D’ … ‘Y’ < ‘Z’

No guarantee on upper and lower cases -> depended on the encoding skin

ASCII: lowercase > uppercase

‘ ’ < anything (at least alphabeth and number)

PRETENT TO BE A DICTIONARY – unknown caps

C strings