11/06/19

void makeUpperCase(string& s) //C++ string

{

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

{

s[k] = toupper(s[k]);

}

}

void makeUpperCase(char s[]) //C string

{

for (int k = 0; s[k] != ‘\0’; k++)

{

s[k] = toupper(s[k]);

}

}

cout << attendance [2][5] //row 2, column 5

for (int w = 0; w <NWEEKS; w++)

{

int t = 0;

for (int d = 0, d <NDAYS; d++)

t += attendance [w][d];

cout << “The total for week “ << w << “ is “ << t << endl;

}

const string dayNames [NDAYS} = {“Monday”, “Tuesday”, …, “Sunday”};;

//let weekends be 5, 6, 7

int grandTotal = 0;

for (int d = 4; d < NDAYS; d++)

{

int t = 0;

for (int w = 0; w < NWEEKS; w++)

t += attendance [w][d];

cout << “The total for “ << dayNames[d] << “ is “ << t << endl;

grandTotal += t;

}

cout << “Over the course of” << NWEEKS << “ weeks, weekend attendance was “ << grandTotal << endl;

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;

}

double m2 = meanForADay (const int a[][], int nRows, int dayNumber)

{

if (nRows <= 0)

return 0;

int total = 0;

for (int r = 0, r < nRows; r++)

{

total += attendance[r][daysNumber];

}

}

int main ()

{

int attendance [NWEEKS] [NDAYS];

…

double m = computeMean(attendance [2], NDays);

* unable to treat columns as arrays
* Row -> arrays -> column – each int

double m2 = meanForADay (attendance, NWEEKS, day (ex. 4 -> Friday)

}

An array of array -> 2D array

* before passing array -> need to know howm uhc time mp (how many integers / data

int multiplexAttendance {NEWEEKS}[NDAYS] ]16]; //16-screen complex

void f(omt b[] [NDAYS] 16;

int chainAttendance [NWEEKS] [NDAYS] [10] [16]

string cppstrings[5] = { “aaa’, ‘bbb’, ccccc’, ‘’, “eeee”);

Each row -> array holding C string

const int MAXWORDLENGTH = 6;

//C String

int findFirstOfLength(char a[][MAXWORDLENGTH + 1], int nRows, int targetLnegth)

{

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

{

if (strlen(a[r]) == targetLength)

return r;

}

return -1;

}

//C++ String

int findFirstOfLength(string a[], int nRows, int targetLnegth)

{

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

{

if (a[r].size() == targetLength)

return r;

}

return -1;

}

int main ()

{

const int MAXPETS = 5;

char pets [MAXPETS] [MAXWORDLENGTH + 1] = {‘cat”, “mouse”, “eel”, “ferret”, “horse”};

cout << findFirstOfLength(pets, MAXPETS, 5)

}

Project 5 -> C string

Pointers:

* Another way to implement passing by reference
* Traverse arrays
* Manipulate dynamic storage
* Represent relationships in data structures

#include <cmath>

using namespace std;

void polarToCartesian (double rho, double theta, double\* xx, double\* yy);

int main ()

{

double r;

double angle;

… get values for r and angle

double x;

double y;

polarToCartesian(r, angle, &x, &y);

…

}

void polarToCartesian (double rho, double theta, double\* xx, double\* yy);

{

\*xx = rho \* cos(theta);

\*yy = rho \* sin(theta);

}

double& means reference-to-double or another-name-for-some-double

//only for C++, which has reference type (not C)

main: r [value], angle [value], x [trash], y[trash]

polarToCortesian:

double\* means point-to-double or address-of-some-double

&x means generate a pointer to x or address of x

\*xx means the object that xx points to or follow the pointer xx

Point requires extra declaration to follow the pointers -> worse than reference-type

(At least in C++) – Pointer itself is passed by value – pass by ref ~ changing values

Pointer Manipulation

double a = 3.2;

double b = 5.1;

Tips: when using points, draw pictures -> forgetting what is pointing to what

double\* p = &a; (generate point a and putting it into p)

//double\* q = b; //ERROR (diff types; one is pointer and the other is double)

double\*q = &b;

double c = a; (c = 3.2)

//double d = p; //ERROR (diff types; one is double and one is pointer (arrow to double))

double d = \*p; (double to double, which is what p is pointing to, which is a in this case)

//p = b; //ERROR! (setting pointer to double)

p = &b; OR \*p = b; (DIFFERENT MEANING: first one point to b -> p is still a pointer pointing to b, whereas the other change a, which is value of what p was pointing to, to b;

1. p = &b;

* a = original (3.2), p is pointing to b now (5.1)

1. \*p = b;

* a = b (5.1), p is still pointing to a

When compilation error -> make sure which error is trying to be fixed (what is being changed? the value/ variable itself or where the pointer is changing)

Continue 1.)

p = &b;

\*p += 4; (b is adding 4)

int k = 7;

//p = &k; //ERROR (a pointer to an int cannot be a pointer to a double) – diff pointer types

int \*z = &k;

11/13/19

#include <iostream>

#include <cmath>

void polarToCartesian (double rho, double theta, double\* xx, double\* yy);

int main ()

{

}

void polarToCartesian (double rho, double theta, double\* xx, double\* yy)

{

\*xx = rho\*cos(theta);

\*yy = rho \*sin(theta);

} …

Continued – Statement/Error Examples for Pointers

Right now:  
int k = 7; int\* z = &k;

double a = 3.2;

double b = 9.1; double\* p = &b;

double c = 3.2;

double d = 3.2;

cout << (k\*b); //legal – int\*double (9.1\*7=63.7)

cout << (k\*p); //illegal – (\* binary operator) cannot operate on pointer with int;

cout << (k\*\*p); //legal (k multiplies by the object p is pointing to)

cout << (\*z\*\*p) //legal -> cout << (\*z \* \*p); //amount of space doesn’t matter

double\* q;

\*q = 4; //able to compile, but execution fails because q is not initialized

-> undefined behavior (following bad pointers – value 0 is replacing an unimportant location -> may or may not work) //under g31 -> recognize and crush the program

double\* r = &a; //(r points to a with value 3.2)

\*r = b; //legal //change a to the value to b (a = 9.1);

if (p == r) //false (pointers pointing to different object, even though same values (9.1))

q = p; //legal, q now is pointing to p, whose address is pointing to b (is q is now pointing to b as well)

if (p == q) //TRUE – pointing to the same place

if (…p… == …r… ) // true -> if (\*p == \*r) // true – testing the values, doubles (SAME)

if (&a == &b) //false – different pointers

Other: pointer to char -> print out the whole array (in C string)

Ex: char s[10] = “Hello”; cout << s; (print “Hello”)

Print pointer

cout << \*p; //write 9.1;

cout << p; //write pointer (the address to b) – different libraries

Pointer to array:

const int MAXSIZE = 5;

double da[MAXSIZE];

int k ;

double\* dp;

* da with 5 elements of double (0-4)

for (k = 0; k < MAXSIZE; k++)

{

da[k] = 3.6;

}

OR

8 bits (1 byte) long (double) -> da[0] at 1000-1007, da[1] at 1008-1015, da[2] at 1016-1023, da[3] at 1024-1031, da[4] at 1032-1039, da[5] //not real, just after// at 1032-1039

for (dp = &da[0]; dp <; ...;) //setting the pointer to point at each item of the array

{

\*dp = 3.6; //the pointer dp finds the location 1000, represented by da[0], and set it to 3.6;

}

OR

dp = &da[0] = (loc)1000;

\*dp = 3.6;

\*(&da[0]); //dp = &da[0])

da[0] = 3.6; //ex. \*&x = &\*x = x;

dp++;

dp += 1;

dp = dp + 1; adding an int to the pointer – only well-defined when the pointer is pointing to an array as the location of the array is well-specified;

&a[i] + j = &a[i+j];

dp++; //same as &da[i++];

Pictorial: the pointer is just pointing to the next element

Machine language: dp++ is the same as adding 8 to the location (as double is 8 bytes long)

for (dp = &da[0]; dp < &da[MAXSIZE]; dp ++)

{

\*dp = 3.6;

}

Another ex: 1000 ft -> 4 yard (different units) similar to machine language and pictorial

* conversion 4 yard -> 12 ft

C++ language level: pointer to an array – when adding the pointer, adding number of object sizes (8 bytes for double)

DO NOT add 8 to dp (at the language level, not machine level) – complier takes account

It is possible to point to the non-existing

&a[i] < &a[j] -> i < j //(if the pointer is pointing to an element earlier in the array, it is smaller than the pointer pointing to an element later in the array)

-possible operator (<. >, ==, !=, <=, >=)

SAME RESULT:

const int MAXSIZE = 5;

double da[MAXSIZE];

int k ;

double\* dp;

for (k = 0; k < MAXSIZE; k++)

{

da[k] = 3.6;

}

for (dp = &da[0]; dp < &da[MAXSIZE]; dp ++)

{

\*dp = 3.6;

}

* Similar to comparison (comparing two pointers in the same array, else undefined)

Simplified

* array a => &a[0] – pointing to the first element of the array
* &da[MAXSIZE+0] = &da[0] + MAXSIZE = a + MAXSIZE;

for (dp = da; dp < da + MAXSIZE ; dp++)

{

\*dp = 3.6;

}

* More efficient; dp++
* Before: k++ is less efficient (finding the actual location again and again)
  + Transformed – using pointer style (same now) compare to subscript
  + C++ motivates to use pointer style – similar to other complexity

string a[6] = {“rat”, “pig”, “goat”, “zebra”, “pig”, “rat”};

int n = lookup(a, 6, “zebra”); //3 (assignment 4)

loopup(&b[0], 6, “zebra”);

Parameter for lookup: lookup(const int a[], int n; string s)

* Not an array (const int a[]), just a pointer to an array = const int a)
* In parameter: by definition – passing int a[] is the same as passing int a);

int k = lookup(&b[2], 3, “zebra”); //legal //new array starting from “goat” – return 1

//In parameter and condition, array -> referring the pointer of the array

a[i] = a + i; ???

a -> &a[0]

p[i] => \*(p+i)

a[1]

\*(a+1)