Lecture 1

Memory

CPU (Central processing Unit)

I/O devices

Input device (bring information from outside to the computer)

Output device

Integers are the only thing that can be held in computer

How to represent 31.43?

1. Store the integer and the fraction separately
2. One integer for the number, one integer for the scientific number
3. Numerator and denominator

How to represent Hello!

Link character to integers

ASCII:

H 72

e 101

space 32

! 33

5 53

Unicode (universal coding system)

Lecture 2

Project

Start early!!!

Develop incrementally!!

Read what we write!!

Von Neumann architecture

Machine language

0: 21004

1: 11005

2: 22006

3: 99999

4: 00042

5: 00013

6: 37285 🡪 00055

…

15: 85054

…

ALU (arithmetic logic unit)

Accumulator: 00042

Instruction Counter: 001

21 004

Operation code memory address

Accumulator: 00055

Instruction Counter: 002

11 005

Operation code memory address

Accumulator: 00055

Instruction Counter: 003

22 006

Operation code memory address

Accumulator: 00055

Instruction Counter: 003

99 999

Operation code memory address

21: copy the number at the indicated memory address into the accumulator

11: add the number at the indicated memory address to the accumulator

22: copy the number in the accumulator to the indicated memory location

99: halt (address par ignored)

Lecture 3

Assembly language

Program is assembled into machine language by a assembler

LOAD PRICE

ADD FEES

STORE TOTAL

HALT

PRICE DATA 42

FEES DATA 13

TOTAL DATA

FORTRAN

integer price = 42

integer fees =13

integer total =price + fees

high-level language

program is compiled into machine language by a compiler

C C++ objective-C Java (has origin in C++) c# (microsoft) swift

Ingredients

Cake recipe🡪 E-to-Ch robot1 🡪 cooking robot 2 (recipe has to be in Chinese) 🡪 yummy cake

(in English)

beat 3 eggs in a mixing bowl

in eggs mixing beat bowl 3 a compilation error/ syntax error (translation is no completed)

beat 3333 eggs in a mixing bowl logic error / runtime error (error during the execution of the program)

logic error can make

#include <iostream> // enable to use the library

using namespace std;// be able to leave std in later part

int main()

{

cout << "Hello" << endl;

}1

*declaration*

*variable*

*letter letter/digit/underscore*

*income2014 valid*

*hours\_worked*

*hoursworked*

double about 10 to the -308 to 10 to the 308

about 15 significant digits

int -2 billion to 2 billion (take up less memory, human readability)

if a number does not have a fraction make it int, otherwise make it double

*types*

Lecture 4

Pay attention to the space in the expression

“payrate” is different from “payrate ”

Arithmetic expression

\* / %

+ -

Multiplication and division has higher precedence than addition and subtraction

Parentheses has higher precedence than multiplication and division

For equal operation, associativity is from left to right

Explicit operators are required (3+4)\* (7-2)

27/ 3\*3 = 27

Certain operation can only apply to certain variables. For example, no arithmetic expression for string.

14.3/ 5.0 --> 2.86

14.3/ 5 🡪 2.86

14 / 5.0 🡪 2.8

if at least one number in expression is double, the result will be double

14/ 5 🡪 2

14 % 5 🡪 4 (modulus, remainder)

cout.setf(ios::fixed);

cout.precision(2);

change the decimal place

100🡪 100.00 123.456🡪 123.46 (round up)

A declaration is of the form

type variableName;

type variableName = expression

anything in the same line after // is ignored

or anything between

double x = 3.1 + 14/5; // x is 5.1, not 5.8

int a = 10;

int b = a\*a;

int c= 25/(b-100); // undefined behavior. Most likely to crash. This is a run time error

double d;

double e= 2\*d

cout << e; //

int f = 1000;

int g = f \* f \* f;

int h = g \* f; // exceed the maximum value that int can store (2 billion), it usually just take the lower order digits

string is not a declaration built in c++

if string is not defined, it start with the empty value. (it’s already initialized in the library)

Lecture 5

Only need cin.ignore if there is string after number

string s;

int I;

cin >> I;

geline (cin,s)

Program does os writes os holds available to program

getline(cin,s)

you type x x x program waits

you type y y x y

you type BS BS space BS x

you type Z z z

you type Enter CR LF x z newline

getline executes

(empty) s gets “xz”

cin >> I;

you type 3 3 3

you type 2 2 3 2

you type Enter CR LF 3 2 newline

cin >> I executes

newline (is not consumed) sets I to 32

getline(cin,s) (empty) s gets “” (length of the string is 0)

cin.ignore(10000, ’\n’); (empty)

getline(cin, s); getline waits

if (condition)

statementIfTrue

else

statementIfFalse

oneThing > anotherThing

oneThing >= anotherThing

onething <= anotherThing

oneThing != anotherThing not equal

oneThing == anotherThing is equal to

check the great, smaller and equal case

more than (greater)

at least (greater or equal)

less than (smaller)

no more than, at most (smaller or equal)

{ stmt; stmt; stmt; } a compound statement / a block

if (x>0)

cout << “sjel!” << endl;

else

{

getline (cin,s);

cout << “flksdjelkj” << endl;

}

if (condition)

statementIfTrue

cout << “What is your hourly rate of pay”

if (payRate >= 12.00)

double withHoldingRate = 0.10 // if claim a variable in branch of if statement, the variable can only be used in that branch

double withHoldingRate = 0.05 // wrong!

A declaration of the form the variable did not already exist before this

type variableName;

or type variableName = expression;

An assignment is of the form the variable has been previously declared

variableName = expression

int m = 2;

int n = 3;

…

n = 4\*m; // replace what was there before. n is now 8

…

n = 2\*n; // n is now 16

int a = 3;

int b = a + 5; // b is now 8

a = 4; // b is still 8

b = a + 40; // b is now 44

a = 5; // b is still 44

if (citizenship == “US”)

if (age >= 18)

cout << “You can vote in U.S. elections” << endl;

else

cout << “You are not a U.S. citizen” << endl; // else is paired up with the nearest if

if (citizenship == “US”)

{if (age >= 18)

cout << “You can vote in U.S. elections” << endl;

}

else

cout << “You are not a U.S. citizen” << endl; // use the curly brace to fix

if (citizenship ==”US” || citizenship == “Canada”)

cout << ”The country code is 1” << endl;

combine conditions

condition || condition “or” true if either condition is true, or both

condition && condition “and” true if both conditions are true

&& has higher precedence than ||

if (citizenship == “US” && age >= 18)

cout << “You can vote in U.S. elections” << endl;

if (roll == 2 || roll == 3 || roll == 12)

cout << “You lose!” << endl;

if (citizenship == ”US” && (age < 18 || age >=65))

Lecture 6

“magic numbers” things that are subject to changes

if ( age >= 18 && age <= 20)

if (age <= age <= 20) // Error! Won’t compile!

if (age == 18 || age == 19 || age == 20)

if (age == 18 || 19 || 20) // will compile, but doesn’t do what you think it does .

if (rating < 1 ||| rating > 10)

… bad rating …

if ( rating > 1 || rating < 10) // wrong for what I want to do

… good rating …

// if the rating is -15, this statement is true

if ( rating >= 1 && rating <= 10)

De Morgan’s laws

not (A or B) 🡺 (not A) and (not B)

not (A and B) 🡺 (not A) or (not B)

not (a > b) 🡺 a <= b

not (a >= b) 🡺 a < b

not (a != b) 🡺 a == b

int n = 17

cout << “n is ” << n << endl;

if (n = 17) // wrong

cout << “n is zero” << endl;

else

cout << “n is not zero; n is ” << n << endl;

int a = 10;

int b = 20;

int c = 30;

c = (a = b); // Legal. Sets a to 20, sets c to 20

c = a = b; // Legal. Sets a to 20, sets c to 20

In a context where a true/ false value is required,

the number 0 is treated as false, and any non-zero number is treated as true

if ladder

if (income < 30000)

cout << “Low”;

else if (income , 80000)

cout << “Middle”;

else if (income < 200000)

cout << “High”;

else

cout << “Very high”

switch (choice)

{

case 1:

…

breaks;

case 2:

case 4:

…

break; // if there were no break. Choice 2 or 4 will do its statement and the

statement of 3 or 5

case 3:

case 5:

…

break;

default:

…

}

…

string city;

getline (cin, city);

switch (city) // Error! Can’t switch on a string

{

case “Los Angeles”

…

break;

}

How many times do you want to be greeted? 3

Hello

Hello

Hello

If (condition)

statement

🡺

while (condition)

statement

n = n+ 7; 🡺 n+= 7;

n = n\*2 ; 🡺 n \*= 2;

n /= 2;

n =n + 1; 🡺 n +=1; 🡺 n++; ++n;

int k = 0;

while (k< nTImes)

{ cout << “Hello ” << endl;

k++;

}

cout << “Good to see you” << endl;

}

// if we don’t write k++; this loop will never terminate

Lecture 7

while (condition)

statement // prefer the while loop

do

statement

while (condition) ; // start off by doing the statement, then check the condition.

int k = 10;

while (k> 0); // semicolon here is not what you want!!! Infinite loop. While Keep checking the condition.

{ cout << k << endl;

k++;

}

//; is an empty statement. It’s a valid statement.

If (citizenship == “us”) ; // semicolon here is not what you want

cout << “You are a U.S. citizen” << endl // the program will always write this out

for (int k = 0; k < nTimes; k++ )

{ …

…

}

for (initialization; stay-in-loop condition; prepare-for-next-step iteration)

statement

“off-by-one error” “fencepost error”

for (int k = 1; k < 10; k ++)

cout << k << endl; // last number to write out is 9

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

cout << k << endl; // run 11 times, last one to write out is 10

for (int n =1; n < 1000 ; n \*= 2)

cout << n << end;

cout << k << endl; // wrong, cant use k outside the loop

int k;

for ( k=0; k<10; k++)

cout << k << endl;

cout << “k has the value” << k << endl; // now we can use k outside the loop

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

{

for (in c = 1; c <= 4; c++)

cout << “\*” << endl;

} // we end up with a vertical line of stars

Int r;

for (r= 1; r <= 3; r++)

{

for (in c = 1; c <= 4; c++)

cout << “\*” ;

cout << endl; }

cout << r << endl;// r=4

Lecture 8 01234

string s = ”Hello”; s: Hello

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

cout << s[k] << endl;

cout << s[5] <<s[-1] <<endl; // undefined behavior!!

Enter some text: Hello, Everyone!

The number of Es (upper and lower case) is 4

t: Hello, Everyone!

Cout << “Enter some text: ”;

String t;

getline (cin,t);

int numberOfEs = 0;

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

{

if ( t[k] == ‘E’|| t[k] == ‘e’) // t[k] is a character

numberOfEs++;

}

cout << “The number of Es (upper and lower case) is ” << numberOfEs << endl;

string s = “Hello”; // s is a string; “Hello” is a string literal

char C = s[1]; // c is a char (initialized to lower case e)

// ‘e’ is a character constant

‘B’ ‘w’ ‘@’ ‘ ‘ ‘4’ ‘\t’//tab character

‘\n’// new line character

‘\’’ // single quote character

‘\\’ // single backslash character

cout << “hi\tthere\nbye\n”// this line contains 13 characters

hi there

bye

// char can only have exactly one character in it

cout << “she said, \”wow!\””

she said, “wow!”

char c = ‘x’ // OK

char d = “x”; // Error! Won’t compile. D is a char, “x” is a string literal

string s =”x”; // OK

string t = ‘x’; // Error! Won’t compile. t is a string, ‘x’ is a char

if (t[k] == ‘E’ || ‘e’) // wrong!! (Bur it will compile and do the wrong thing) ‘e’

// will be treated as true. (only 0 is treated as true)

isdigit(ch) ‘0’ ‘1’ ‘2’ … ‘9’

islower(ch) ‘a’ ‘b’ ‘c’ … ‘z’

isupper(ch) ‘A’ ‘B’ ‘C’

isalpha(ch) ‘A’ … ‘Z’ ‘a’… ‘z’

if (! Isdigit(num[k])) //is num[k] is NOT a digit

! A not

A && B and

A||B or

if (! (x>=1 && x<=10))

if ( x < 1 || x > 10)// not necessary

int main()

{

cout << "Enter a phone number: ";

string num;

getline (cin, num);

int numberOfDigits = 0;

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

{

if (isdigit(num[k]))

numberOfDigits++;

}

if (numberOfDigits!= 10)

cout << "A phone number must have 10 digits." << endl;

}

Void greet(int nTimes, string msg);// function prototype,

//Highlighted part is not necessary

int squre(int k);

int main()

{ …

…

greet(square(3),”Hello”); <---

…

…

int a=4;

great (2 \* square(a+3), “Ni hao”);

…

…

string s;

getline(cin, s); // user types Namaste

int n;

cin>> n;

greet(n,s);

…

…

int square(int k)

{

reture k \* k;

}

}

void greet(int nTims, string msg) nTimes: 9 msg; Hello

{

for (int k = 1; k <= nTimes; k++)

cout << "Hello" << endl;

}

Lecture 8

A && B / A || B🡺 if A is false, the result is false; otherwise, the result is B. If the first condition is false, the second condition won’t be evaluated. (in the case of B is undefined behavior)

If (b != 0 && d !=0 && a/b + c/d > 5)

If (isdigit(s[0]) && s.size() != 0)

The first part is undefined. This is checking in the wrong order.

double x = greet(….); // won’t compile ! greet doesn’t return a value. Greet is a void function

Square(5); // compiles, but is useless

a+7; //compiles, but is useless.

If (s.size() != 0)

S[0] = toupper(s[0]); // lower case letters are turned in to upper letters. Others letters will give back the original letters

if (tolower(s[k]) == ‘e’)

if (s[k] == ‘E’ || s[k] == ‘e’)

s = toupper(s);// Error! Won’t compile. S is a string, not a char

toupper(s[0]); // compiles, but is useless.

Enter a phone number: (310) 825-4321

The digits in the number 3108254321

Int main()

{

cout << “Enter a phone number: ”;

string phone;

getline(cin, phone);

if()

cout << “The digits in the number are” << …… << endl;

else

cout << “A phone number must have 10 digits.” << endl;

}

A function that tests a property of something and returns a bool is often called “predicate”

isdigit(s[k])

isValidPhoneNumber(phone)

hasMoreThanTwoPrimeFactors(n)

livesIn(personName, city) // returns true if the person lives in that city

makesMoreThan(employee1, employee2)

checkEmployees(emplyee1, employee2)// bad and vague name

#include <cmath>

using namespae 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);

double x2;

polarToCartesian(2\*r, angle, X2, y);

}

void polarToCartesian(double rho, double theta, double& xx, double& yy)

{

xx = rho \* cos (theta)

yy = rho \* sin (theta)

}

“passing by value” 🡺 copy the arguments into new variables for the parameters

“passing by reference” 🡺 no copy is made; we are just using another name for already existing variable

Lecture 9

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;

…

}

int exchange(int x, int y) // WRONG

{

int t = x;

x = y;

y = t;

}

void exchange(int& x, int& y)

{

int t = x;

x = y;

y = t;

}

int 🡺 int

int& 🡺 reference-to-int

what should I censor? My SSN is 123-45-6789

My SSN is \*\*\*-\*\*-\*\*\*\*

Void censorDigis(string & s)

{

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

{

if(isdigit(s[k]))

s[k]= ‘\*’;

}

}

string censored(string s)

{

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

{

if(isdigit(s[k]))

s[k]= ‘\*’;

}

return s;

}

int main ()

{

cout << “what should I censor?”;

string msg;

getline(cin.msg);

censorDigits(msg); // msg is changed

cout << msg << endl;

cout << “what should I censor?”;

string msgs;

getline(cin.msgs);

cout << censored(msg2) << endl; // msg2 is unchanged

string msg3 = censored(msg2); // msg2 is unchanged

msg2 = censored(msg2); // now msg2 is changed

}

first method is less flexible, since it change the value of the original string. The second method returns a new string, while leave the original string unchanged

string phone;

cout << “Enter a phone number: ”;

getline(cin, phone);

while ( ! isValidPhoneNumber(phone))

{

cout << “A phone number must have 10 digits; try again.” << endl;

cout << “Enter a phone number: ”;

getline(cin,phone);

}

cout << “The digits in the number are” << digitsOf(phone) << endl;

string phone;

while (true)

{

cout << “Enter a phone number: ”;

getline(cin.phone);

if(isValidPhoneNumber(phone))

break; // breaks out of the nearest enclosing loop or switch

cout << “A phone number must have 10 digits; try again.” << endl;

}

cout << “The digits in the number are” << digitsOf(phone) << endl;

“N-and-a-half-times loop”

string phone;

For (;;)

{

cout << “Enter a phone number: ”;

getline(cin.phone);

if(isValidPhoneNumber(phone))

break; // breaks out of the nearest enclosing loop or switch

cout << “A phone number must have 10 digits; try again.” << endl;

}

cout << “The digits in the number are” << digitsOf(phone) << endl;

“N-and-a-half-times loop”

for(…;…;…)

{

…

if(….)

{

…

if (…)

{

…short thing …

continue;// jump back to the for loop

}

…

… the rest of the iteration

}

}

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

int k = ‘L’; // 76 if ASCII, 211 if EBCDIC

‘a’ … ‘z’ are in order. Aren’t guaranteed it’s consecutive

‘A’ … ‘Z’ are in order. Aren’t guaranteed it’s consecutive

‘0’ … ‘9’ are in order and consecutive

ch++; // Now 77; ‘M’ if ASCII, ‘(’ in EBCDIC

k = 77;

double x = 3.7;

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

// if ASCII, 51 46 55

// if EBCDIC, 243 75 247

cout << k; // calls the function for int: writes ‘7’ ‘7’

// if ASCII, 55 55

// if EBCDIC, 247 247

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

// if ASCII, this is 77

// if EBCDIC, this is 77

‘7’

7

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

{

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

return false;

if (m == 1 || m == 3 || …)

return d<= 31; // function return true of false depending on the value of the expression

if (m == 4 || m == 6 || …)

return d < = 30;

……

}

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

{

…

const int MONTHS\_IN\_A\_SEASON = 3;

const int NMONTHS = 12;

const int daysInMonth[NMONTHS] = {

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

31, 31, 30, 31, 30, 31

};

if (m < 1 || m > 12 || d < 1 || d > daysInMonth[m-1])

return false;

…

daysINMONTHS[3] = 42; // won’t compile! daysINMONTHS[3] is const

}

0 1 2 3 4 5 6 7 8 9 10 11

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

daysInMonth[k] when k is out of range is undefined behavior

switch(m)

{

case 1: cout << “January” << endl; break;

case 2:

…

}

cout << …… m…… << endl;

string monthNames[NMONTHS] = {

“January”, “February”, …

};

cout << “Here are the months with 31 days:” << endl;

for (int k = 0; k < NMONTHS; k++) // k <= NMONTHS -1 also works

{

if (daysInMonth[k] == 31)

cout << monthNames[k] << endl;

}

…… daysInMonth.size()… // NO!!!!

…… daysInMonth.length …… // NO!!!

const int MAX\_NUMBER\_OF\_SCORES = 1000;

int scores[MAX\_NUMBER\_OF\_SCORES];

int total = 0;

int nScores = 0;

cout << “Enter the scores (negative when done):” << endl;

for(;;)

{

int s;

cin >> s;

if (s < 0)

break;

if (s > 100)

{

cout << “A score must be in the range 0 through 100; score ignored.”

<< endl;

continue;

}

total += s;

scores[nScores] = s;

nScores++;

}

if (nScore == 0)

cout << “There were no scores, so no statisctics” << endl;

else

{

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

cout << “The mean is ” << mean << endl; // 1.0\*total works too

int sumOfSquares = 0;

for (int k =; k < nScores; k++)

{

double diff = scores[k] – mean;

sumOfSquares += diff \* diff;

}

cout << “The std. deviation is ” << sqrt(sumOfSquares/nScores) << endl;

}

double computeMean( int a[] , int n)// the second parameter tells us how many items we want to look at

{

if (n <= 0)

return 0.0;

int total = 0;

for (int k = 0; k < …; k++)

total += a[k];

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

}

int main()

{

double scores[1000];

int nScore = 0;

… fill up all or part of the array

double m = computeMean(……);

…

int stuff[100];

… fill up the entire array

double m2 = computeMean(stuff, 100);

}

Lecture 10

Const int MAX\_NUMBER\_OF\_SCORES = 1000;

Int scores[MAX\_NUMBER\_OF\_SCORES];

Int total = 0;

Cout << “Enter the scores (negative when done):” << endl;

For (;;)

{

int s;

cin >> s;

if (s < 0)

break;

if (nScores == MAX\_NUMBER\_OF\_SCORES)

{

cout << “I can handle only ” << MAX\_NUMBER\_OF\_SCORES << “scores!” << endl;

cout << “Continuing with just the first ” << MAX\_NUMBER\_OF\_SCORES << “values.” << endl;

break;

}

if (s > 100)

{

cout << “A score must be in the range 0 through 100; score ignored.” << endl;

continue;

}

total += s;

scores[nScores] = s;

nScores++;

}

if (nScores == 0)

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

else

{

double mean

}

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

{

if (n <= 0)

return 0.0

int total = 0;

setAll (a, n, 42) // Error! Won’t compile. Can’t pass it to a function that modify

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

total += a[k];

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

}

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

{

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

a[k] = value;

}

allowed

const array const function

non- const array const function

non- const array non-const function

not allowed

const array non-const function

compiler solely check on the type. Even if a function does not modify, it is considered as modifying function if it does not have const.

const int NWEEKS = 5;

const int NDAYS = 7;

row col

int attendance[NWEEKS][NDAYS];

cout << attendance[2][5];

//for each week (row) of the array,

add up all the elements in that row

write the total

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

{

int total = 0;

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

total += attendance {w}{d};

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

}

//start grand total at 0

//for each weekend day,

add up the attendances for that day over all weeks

write that total

write the grand total

int grandTotal = 0;

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

{

int total = 0;

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

total += attendance[w][d];

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

grandTotal += total;

}

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

for (int p = 0; p < 3; p++)

{

for (int h = 0; h < 3; h++)

…. Attendance[h][p] …

}

M T W

0 1 2 3 4 5 6

0

1

2

3

4

5

6

const int NWEEKS = 5;

cons tint NDAYS = 7;

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

double meanForADay ()

int main()

{

int attendance[NWEEKS][NDAYS];

double meanwed = meanForADay (attendance, NWEEKS, 2 /\* Wed \*/)

}

double meanForADay ( int a[][NDAYS], int nRows, int dayNumber)

{

if (nRows <= 0)

return 0.0;

int total = 0;

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

total += a[w][dayNumber];

return static\_cast<double>(total) / nRows;

}

when passing an array to a function, when declaring the array parameter

1. leave the first pair of square brackets empty
2. Supply the actual declared size for the remaining dimensions

Int multiplexAttendance [5][7][16];

Void g(int a[][7][16])

string s1 = “hello” ;

string s2 = “help” ;

string s3 = “helping” ;

string s4 = “hElp”;

s1 < s2

s2 < s4

if (s2 < s4) // false if ASCII (‘e’ > ‘E’), true if EBCDIC (‘e’ < ‘E’)

C++ strings

#include <string>

using namespace std;

string t = “hello”;

string s; // empty string

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

cout << t[k] << endl;

cout << t;

getline(cin, s);

s = t; // sets s to hello

s += “!!!”; // set s to hello!!!

If (t < s)

Lecture 11

String library is only usable in C++

C strings

#include<cstring>

Char t[10]

0 1 2 3 4 5 6 7 8 9

‘H’ ‘e’ ‘l’ ‘l’ ‘o’ ‘\0’

Zero byte

Never look beyond zero byte

Char t[10] = { ‘H’, ‘e’, ‘l’ , ‘l’ , ‘o’ , ‘\0’};

Char t[10] = “Hello”; // 6 characters, \0 is implied. It can hold 9 interesting characters at most

char s[50]; // s is uninitialized

char s[50] = “”;

for (int k = 0; t[k] != ‘\0’; k++) // to visit every single character in the string

cout << t[k] << endl;

cout << t; // write all the characters up to but not including the single byte

cin.getline(s, 50); // counting zero byte, it is safe to store 50 characters

cout << strlen(t); // writes 5, can’t call size on c string.

//If there is no zero byte, this would cause strlen keep searching until it find zero byte. It would compile

s = t; // won’t compile! Can not assign to an array!

strcpy(s,t); // strcpy(destination, source) if there are elements after the zero byte, they are unchanged

char tooShort[5];

strcpy(tooShort, t); // undefined behavior when trying to assign t[10] to tooShort[5] !

strcpy(tooShort, “wow”); // Ok, need only 4 positions (3 plus the zero byte)

s += “!!!”; // wrong

strcat(s, “!!!”); // Now s is “Hello !!!” have to make sure there is enough room

//if (t < s) // will compile, but will not do what you want!!!

If (strcmp(t,s) < 0)

strcmp(a, b)

negative if a comes before b

1. if a equals b

positive if a comes after b

C++ strings: a OP b

C strings: strcmp(a,b) OP 0

Does a come before b?

Wrong: if (a<b) // will compile for C strings, but not do what you want!!!

Right: if (strcmp(a,b) < 0)

Is a equal to b?

Wrong : if (strcmp(a,b)) // compiles, if a and b are equal false, otherwise true

Right: if (strcmp(a,b) == 0)

void makeUpperCase(string& s)

{

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

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

}

void makeUpperCase(char s[])

{

for (int k = 0; s[k] != ‘\0’; k++)

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

}

const int MAX\_NAME\_LEN = 6;

int tally(char a[][MAX\_NAME\_LEN+1], int n, int targetLength );

int main()

{

const int MAXPETS = 5

char pets[MAXPETS][ MAX\_NAME\_LEN+1] = {

“cat”, “mouse”, “eel”, “”ferret”, “horse”

};

cout << tally(pets, MAXPETS, 5); // writes 2 because of mouse horse

}

int tally(char a[][MAX\_NAME\_LEN+1], int n, int targetLength)

{

int total = 0;

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

{

if (strlen(a[k]) == targetLength) // calls strlen on a C string, a whole row of the array

total++;

}

return total;

}

pointers :

Another way to implement passing by reference

Another way to traverse arrays

Manipulate dynamic storage

Represent relationships in data structures

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);

}

As the name of a type:

double double

double& reference to double

double\* pointer to double

In an expression:

&x means “generate a pointer to x” // unrelated to the other

\*p means “the object that p points to” “follow the pointer p”

c only have pointers, but not reference

a : 3.2 b : 5.1 c: 3.2 d: 3.2 k: 2

a : 5.1 b: 9.1

a : 9.1

double a = 3.2;

double b = 5.1;

double\* p = &a;

double\* q = 7.6; // q can only hold a pointer to a double. Error won’t compile

double c = a;

double d = p; // illegal, a pointer is not a number. Won’t compile

double d = \*p; // \*p the object p points to

p = b; // Error!

\*p = b; // assigning one double to another

p = &b; // assign one pointer to another

\*p += 4; // legal

int k = 2;

p = &k; // Error!

Int \* z = &k;

Cout << (k \* p); // Error! Won’t compile

Cout << (k \* \*p); // writes 18.2, int times a double

Cout << ((\*z) \* (\*p)); // writes 18.2

double\* q;

\*q = 4; // this will compile, but q has not been initialized. However, this will cause a runtime error. Undefined behavior

p = q;

double \*r = &a;

if (p == r) // false comparing pointers

cout << “Hello”;

if (p == q) // true comparing pointers

if ( \*P == \* q) // true comparing doubles

if (p == &b) // true

if (\*p == b) // true

const int MAXSIZE = 5;

double da[MAXSIZE];

int k;

double\* dp;

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

da[k] = 3.6;

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

\*dp = 3.6; // dp tells the address where to put the 3.6

&a[0]

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

\*dp = 3.6;

&da[5]

&da[0 + 5]

&da[0] + 5

da + 5// a pointer to element 5

\*&x 🡺 x

\*dp = 3.6;

\*(&da[0]) = 3.6;

da[0] = 3.6;

&a[i] + j 🡺 & a [i + j]

Const int MAXSIZE = 5;

Double da[MAXSIZE];

Int k;

Double\* dp;

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

da[k] =3.6;

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

\*dp = 3.6;

dp gets some value

int k = dp – da // pointer arithmetic rules perform the conversion from bytes automatically

dp++;

dp = dp + 1;

dp = &da[0] + 1;

dp = &da[0 + 1];

dp = &da[1];

&a[i] < &a[j] 🡺 i < j

dp < da + 5

dp < &da[0] + 5

dp < &da[0+5]

dp < &da[5]

&da[0] < &da[5]

0 < 5

true

\*dp = 3.6

\*(&da[0]) = 3.6

da[0] = 3.6

dp ++

a = &a[0]

&a[i] - &a[j] => I – j

string\* a

void appendToAll(string a[], int n, string value) // you are passing a pointer

{

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

a[k] += value; //

}

int main()

{

string stuff[5] ={“Hello”, “Goodbye”, “wow”, “Yes”, “No”};

appendToAll( stuff, s ,”!!!” ); // pointer to element 0, stuff => &stuff[0]

appendToAll(stuff, 3 , “!!!”);

appendToAll(&stuff[2], 3 , “!!!”);

}

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

a[2]

\*(a + 2)

\* (&stuff[0] + 2)

\*(&stuff[0 + 2])

\*(&stuff[2])

stuff[2]

appendToAll(&stuff[2], 3 , “!!!”);

a[1]

\*(a+1)

\*(&stuff[2]+1)

\*(&stuff[2+1])

\*(&stuff[3])

stuff[3]

a pointer to one array minus pointer to another array will compile, but will result in an undefined behavior

double\* findFirstNegative(double a[], int n)

{

for (double\* dp = a; dp < a + n ; dp++ )

{

if (\*dp < 0)

return dp ;

}

return nullptr;

}

int main()

{

double da[5];

…

double\* pfn = findFirstNegative(da, 5);

if (pfn == nullptr)

cout << “There are no negative values in the array” << endl;

else

{

cout << “The first negative value is ” <<\*pfn << endl;

cout << “It’s at element number” << pfn - da << endl;

}

}

C++11 nullptr

C, C++ NULL the integer constant 0 in a context where a pointer is required

Struct Employee // convention: use the capital letter for type introduced

{

string name; // data members, fields, attributes, instance variables

double salary;

int age;

}; // Don’t forget semicolon!!!

Int main()

{

Employee e1; // name will be empty string

Employee e2;

e1.name = “Fred”;

e1.salary = 50000;

e1.age = 42;

e1.age++;

cout << “Enter a name: ”;

getline(cin, e2.name);

cout << “Enter a salary: ”

cin >> e2.salary;

Employee company[100];

company[2].name = “Ricky” // name member of a employee

for (int k = 0; k != company[2].name. size(); k++)

cout << company[2].name[k]<< endl;

}

an object of some struct type . the name of a member of that type

a pointer to an object of some struct type -> the name of a member of that struct

the name of some struct type :: the name of a member of that type

p -> m means (\*p).m

void target :: init ()

{

this -> pos = 0;

}

Struct Employee

{

string name;

double salary;

int age;

};

void printPaycheck (const Employee& e)

{

cout << “Pay to the order of ” << e.name<< “ the amt $” << e.salary/12 << endl;

}

void celebrateBirthday(Employee& e)

{

e.age++;

}

void celebrateBirthday2(Employee\* ep)

{

ep -> age++; // same as (\*ep).age++;

}

double tottalPayroll(const Employee emps[], int n)

{

double total = 0;

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

total += emps[k].salary;

return total;

}

int main()

{

Employee company[100];

Int nEmployees = 0;

… read in some info, fill array, set nEmployees

print Paycheck(company[1]);

celebrateBirthday(company[0]);

celebrateBirthday2(&company[0]);

double t = totalPayroll(company, nEmployees);

…

for (Employee\* ep = company; ep < company + nEmployees; ep++)

cout << ep -> name << endl;

}

If the function is not going to modify the argument:

Pass by value, e.g.

Struct Target

{

public:

void init();

bool move (char dir);

// Invariant:

// history consists only of Rs and Ls

// pos == number of Rs in history minus

number of Ls in history

private:

int pos;

string history;

};

void Target:: init()

{

this -> pos = 0;

this -> history =”0”;

}

bool Target:: move(char dir)

{

switch (dir)

{

case ‘R’:

case ‘r’:

this -> pos++;

break;

case ‘L’:

case ‘l’:

this -> pos--;

break;

default:

return false;

}

this -> history += toupper(dir);

reture true;

}

int target::position()

{

return this -> pos; // this -> can be left off

}

void Target:: replayHistory()

{

for (int k = 0; k != this -> history.size(); k++)

cout << history[k] << endl;

}

int main()

{

Target t;

t.init();

t.move(‘R’);

t.move(‘R’);

t.move(‘L’);

t.pos = 42; // won’t compile!

int n = t.position();

char ch;

…sets ch…

if (! T.move(ch))

… problem …

t.pos++;

t.history +=’R’;

…

Target t2;

t2.init();

}

string s;

…

cout << s.size();

cout.precision(2);