Course Review

This document summarizes lectures from Feb 21 to March 26. Before Feb 21, our lectures covered the following topics: variables, operators, truth table, if statement, switch statement, for loop, while loop, function.

This document covers the following topics: struct, class, binary and hexadecimal, fstream, array, pointer, comparison between pass by value, pass by pointer and pass by reference.

# Struct

structure in c++ is form of aggregation. We have 3 steps to use a struct: declare a struct, create instances of the struct and access/use the attributes of an instance

For example:

// (1) declare a struct named “student”

struct student

{

string name;

int age;

float gpa;

float height;

float weight;

};

The “struct” is like a package and the package has a name student. Defining a “struct student” is just like declaring a new type with five things associate with it.

After declare it, we can create instance of this type.

// (2)create an instance of struct student

student \_s;

And then, we can access the attribute of the instance \_s

// (3) here we print the “age” attribute of instance “\_s”

cout << "\_s.age: " << \_s.age << endl;

# Class

Class is a formal aggregation type in C++. Class permits encapsulation and abstraction. Data and function/method insides the class are called class members. We can use **class visibility specifiers** to make the class member visible outside of the class. We covered two class visibility specifier, one is “private”, the other is “public”. We can’t access private members outside of the class; we can access public members outside of the class.

## Difference between struct and class

The only difference between class and structure is the default visibility is different. **In struct, all the struct members are public; in class, the default visibility specifier is private**. To access the private members in class, we can define some public method to get or set the private members. These public methods are often called getter or setter/mutator.

## 2.2 class constructor

Class constructors should have exactly the same name with the class name and class constructor does not need to have return type.

A class can have more than one constructor, but these constructors must have different argument lists.

## 2.3 code and explanations

#include<iostream>

#include<cstdlib>

#include<ctime>

using namespace std;

class Playa // here we define a class, named Playa

{

private: // private is the visibility specifier, which means we can't access these members outside of the class

// here we define 5 private members. We usually make data members private in code.

string Tag;

int Armorclass;

int HitPoints;

int ManaPoints;

int NumFrags;

public:// public is the visibility specifier, which means we can access these members outside of the class

// here we define 3 public members. We usually make functions/methods members public in code.

Playa(); // This is the constructor of the class. constructor should have exactly the same name as the class name

// we usually use constructor to initialize the data members

void TagSet(string NewTag); // we usually define a function set() to updated the values of data members

string TagGet(); // we usually define a function get() to access the values of data members

// we still can have other functions/methods in the class, as we have seen the following assignments

};

// the following are the definition of the class function

// to define a function of a class, we need scope resolution operator.

// scope resolution operator is ::

Playa::Playa()

{

Tag = "-default-n00b-";

Armorclass = 0;

HitPoints = 0;

ManaPoints = 0;

NumFrags = 0;

}

string Playa::TagGet()

{

return Tag;

}

void Playa::TagSet(string NewTag)

{

Tag = NewTag;

}

int main()

{

int i;

i = 71;

Playa p1; // here we create an instance/object of a class.

// this code means we define a varialbe, whose name is "p1" and whose type is "Playa"

// this code has exactly the same meaning as "int i", where we define a variable i with type int

Playa p2;

// pay special attention to code enclosed between /\* \*/. if we want to assign/update values of the instance of class

// we can't use the code enclosed between /\* \*/.Because data members are private member in class,

// we can't access them outside of class

/\*p1.NumFrags = -1;

cout << "p1.NumFrags: " << p1.NumFrags << endl;

cout << "p2.NumFrags: " << p2.NumFrags << endl;

cout << "p1.tag: " << p1.Tag << endl; // without TagGet() is still private can't access

cout << "p2.tag: " << p2.Tag << endl;

\*/

// instead, if we want to update and access the values of p1.Tag and P2.Tag,

// we need to call the set() function and get() function.

// so here we call TagSet() and TagGet()

p1.TagSet( "123t2sieso");

p2.TagSet( "stirn\_comp\_alit");

cout << "p1.tag: " << p1.TagGet() << endl;

cout << "p2.tag: " << p2.TagGet() << endl;

cout << "Finish" << endl;

}

# Binary and hexadecimal

## representation

to denote binary, we need to two digits, 0 and 1;

to denote decimal, we need 10 digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9;

to denote hexadecimal, we need 16 digits, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

## converse binary number into decimal number

if we converse (1011)2 to decimal,

1\*23 + 0\*22 + 1\*21 +1\*20

= 8 + 0 + 2 + 1 (step 1)

= 11 (step 2)

First step: converse each digit into decimal

For each digit at position n (1, 2,3 ….n) , the formula is Bn \* 2 (n-1), Bn is thedigit at position n;

Second step: sum up decimal numbers together by decimal addition rule.

## converse hexadecimal number into decimal number

to converse hexadecimal into decimal, we follow the same rule:

(D701)16

= D \* 163 + 7 \* 162 + 0 \* 161 + 1 \* 160 (step 1)

= 13 \* 163 + 7 \* 162 + 0 \* 161 + 1 \* 160 (step 1)

= 53248 + 1792 + 0 + 1

= 55041

# fstream

when we want to write to a file/read from a file, we need this library <fstream>. This library has two subclasses. If we want to write something to a file, we use ofstream; if we want to read something from a file, we use ifstram.

## ofstream

Here is an example and explanation of ofstream

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

// Example of write something to a file. we have 4 steps

// 1. create an instance of ofstream class,

ofstream fout;// here the name of the instance is 'fout'

// 2. use open() method of 'fout' to open/create a file

fout.open("test.txt");

// 3. write something to the file use fout

// write a string into file

fout << "It is raning now" << endl; // here we will write "It is raining now" into file test.txt

// more examples of fout, write the values of array val[] to file text.txt

int val [4] = {1,2,3,4};

fout << "val[0]: " << val[0] << endl;

fout << "val[1]: " << val[1] << endl;

fout << "val[2]: " << val[2] << endl;

fout << "val[3]: " << val[3] << endl;

// another more example of fout, write the values of array labs[] to file text.txt

string labs [5] = {"lab1", "lab2", "lab3", "lab4", "lab5"};

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

{

fout << "elements in labs: " << labs[i] << endl;

}

// write another string into the file

fout << "input ended\n";

// 4. close the file

fout.close();

// after we run this code, nothing happens to the computer screen, because we did not cout anything

// but a file named 'test.txt' is created at the same directory, it contains all content in step 3.

}

The output of the code is:

A white background with black text

Description automatically generated

## ifstream

Here is an example and explanation of ifstream

#include<iostream>

#include<fstream>

using namespace std;

int main()

{

// Example of read something from a file. we have four steps

// 1. step create an instance of ifstream class,

// here the name of the instance is 'fin'

ifstream fin;

// 2. use open() method of 'fin' to open a file.

fin.open("test.txt");

// 3. read something from the file

// we need to define some variables to hold the contents that we read from the file

// and then print them out.

// read a single charactor, we use char variable.

char c1;

fin >> c1; // The first time, you use fin >> c1, we read the first character of the file. In this example file,

// The first character is "I"

cout << "Frist character: " << c1 << endl;

char c2;

fin >> c2; // The second time, you use fin >> c2, we read the second character of the file. In this example file,

// The second character is "t"

// Here note fin will remember its last position, and whenever you reuse fin, it will continue

// from its last position.

cout << "Second character: " << c2 << endl; // here output is "t"

// read a single line in the file, we use getline() function

string line1; // here we read the first line

getline(fin, line1);

cout << "First line: " << line1 << endl; // because we have already read "I" and "t" into c1 and c2

// so here the output is " is raning now"

string line2;

getline(fin, line2);

cout << "First line: " << line2 << endl;

// read all the remaining contents in the file

char c;

while((!fin.eof())) // test for the end of the file (eof)

{

fin >> c; // read a charactor

cout << c; // print the charactor. with the while loop, it can print all the characters in the file

}

// 4. close the file

fin.close();

}

# Array

Array is a formal aggregation type in c++. All the elements in an array is homogeneous and are stored continuous.

## array basics

Here is an example of basic example and its explanation

#include<iostream>

#include<fstream>

#include<cstdlib>

using namespace std;

int main(){

int val[5] = {23, 323, 11, 56, 33}; // Here we declare an int array with 5 elements.

// the value of each element is 23, 323, 11, 56, 33 respectively,

// each element has an address, the 5 elements are store continuously

// NOTE THAT: \*\*when we access the array through its name, we access the ADDRESS of its first element\*\*

cout << val << endl; // in my program, the output is 0x7ff7b00f27e0

// This is the ADDRESS of the first element of the array

// in this case, it means the first element in val, i.e. 23 is stored at 0x7ff7b00f27e0

cout << val + 1 << endl; // the address of the second element

// Here Note: to access the address of the array/array element, we have two methods

// one method, use the array name

cout << val << endl; // the address of the first element;

cout << val+1 << endl; // the address of the second element;

for (int i = 0; i < 5; i++) // the addresses of all the elements

{

cout << val + i << endl;

}

// the other method, use the address operator &.

cout << &val[0] << endl; // because val[0] is a value, we can use &val[0] to get its address

// the address of the first element;

cout << &val[1] << endl; // the address of the second element;

for (int i = 0; i < 5; i++) // the addresses of all the elements

{

cout << &val[i] << endl;

}

// Here Note: to access the values of array elements, we have two methods

// one method, use the array indexing

cout << val[0] << endl; // the value of the first element;

cout << val[1] << endl; // the value of the second element;

for (int i = 0; i < 5; i++) // the value of all the elements

{

cout << val[i] << endl;

}

// the other method, use deference \*.

cout << \*val << endl; // because val is an address , we can use \*val to get its value

// the value of the first element;

cout << \*(val+1) << endl; // the value of the second element;

for (int i = 0; i < 5; i++) // the addresses of all the elements

{

cout << \*(val+i) << endl;

}

}

## array as function parameter

Here is an example of array as function parameter

#include<iostream>

#include<ctime>

#include<cstdlib>

using namespace std;

void initArray(int [], int size); //when we use array as function parameter, we need two formal parameters

// int [] is the type of the array

// int size is the size of the arry

// Note: when we use array as function parameter, the function is always pass by reference.

void pirntArray(int [], int size);

void incrementArray(int [], int size, int increNum);

int main()

{

int ary[5];

cout << "ary address: " << ary << endl;

initArray(ary, 5);

pirntArray(ary, 5); // when we call the function, we need to pass the array and its size

// as we known "ary" here is the address of the first element in int ary[5]

// so here, we actually pass the address of the first element in int ary[5]

// that indicates "if we use array as function parameter, when we call the function,

// it is alway 'pass by reference' "

incrementArray(ary, 5, 10); // to demenstrate array is pass by reference again,

// I write a "cout << "a [] address: " << a << endl;" in incrementArray()

// as you can see both

// "cout << "ary address: " << ary << endl;" and "cout << "a [] address: " << a << endl;"

// print the same address.

pirntArray(ary, 5);

}

void initArray(int a[], int size)

{

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

{

a[i] = 2;

}

}

void pirntArray(int a[], int size)

{

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

{

cout << a[i] << endl;

}

}

void incrementArray(int a[], int size, int increNum)

{

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

{

a[i] += increNum;

}

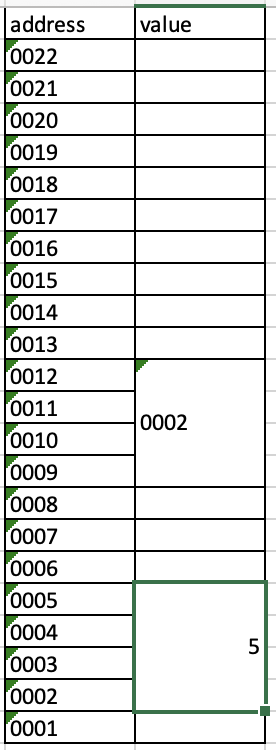
cout << "a [] address: " << a << endl;

}

# Pointer

Pointer is a variable. The value of pointer is an address of another variable.

The figure is a simulation of the pointer.



The block is like a piece of memory.

int i = 5; //the computer finds 4 bytes, from 0002 to 0005

// and write 5 in this place;

int \* p; //the computer finds 4 bytes, from 0009 to 0012

int p = &i; //write the address of variable i in this place

// here write 0002 in this place

\*p; // pointer deference

// computer goes to fetch the value at 0002

// in this case, the value at 0002 is 5

# Comparison between pass the value, pass by pointer, and pass by reference

Here is an example of pass by value, pass by reference and pointer

#include<iostream> // give cin and cout

#include<fstream>

#include<cstdlib>

using namespace std;

void incrementValue(int ); // pass by value

void incrementReference(int &); // pass by reference

void incrementPointer(int \*); // pass by pointer

int main()

{

int a = 10;

cout << "The address of a is: " << &a << endl;

cout << "result of incrementValue(int )" << endl;

incrementValue(a);

cout << "after incrementValue(int ): " << a << endl;

cout << endl;

cout << "result of incrementReference(int &):" << endl;

incrementReference(a);

cout << "after incrementReference(int &): " << a << endl;

cout << endl;

cout << "result of incrementPointer(int \*):" << endl;

incrementPointer(&a);

cout << "after incrementPointer(int \*): " << a << endl;

}

void incrementValue(int i) // pass by value, in main() scope, a = 10

{ // 10 will be copy to incrementValue() scope and assigned to i

i++; // i is changed to 11 in incrementValue() scope

} // but a in main() scope is still 10

void incrementReference(int &i) //pass by reference. in main, we pass a into incrementReference()

{ // & operator get the address of a, and i in incrementReference(int &i) refers to the same address of a in main()

cout << "the value is: " << i << endl; // here i has the same value as a

cout << "the address is: " << &i << endl; // here you can see i has the same address as a

i++; //

}

void incrementPointer(int \*i) // //pass by reference. in main, we pass &a into incrementReference(int \*i)

// note in main, we should pass (&a) into incrementReference()

{

cout << "the value is: " << i << endl; // here the value of i is the address of a

cout << "the address is: " << &i << endl; // here it is the address of pointer \* i

(\*i)++; // in this case the output is 12. because the values at address of a has already been 11 after incrementReference(int &i)

// but it is basic meaning is increment the value by one

}