**Andrew Tran**

**CSCI 1107 Lab Experience Five**

Objectives:

1. Create heterogeneous data types (**struct**s or records)
2. Manipulate **struct**s.
3. Arrays of **struct**s
4. Vectors and the introduction to generic classes using the template mechanism.

**Structs --- Programmer-Defined Data Types**

The construction of a heterogeneous data type in C++ is done through the usage of **struct** keyword. A **struct** is collection of fixed number of components (usually called fields) which are accessed by a variable name representing the struct. The syntax for a struct structure is:

**struct structName {**

**dataType1 identifier1;**

**dataType2 identifier2;**

**…**

**dataTypen identifier;**

**};**

Where:

1. **struct** is a reserved word
2. Datatype – n can either be primitive types, objects, pointers, or other **struct** types.
3. Memory is not allocated until a variable of type structName is declared.
4. The semicolon after the right curly brace is required since this is not a block declaration.

Example:

struct EmployeeType{

string firstName;

string lastName;

string address1;

string address2

string empId;

double salary;

};

The above defines a structure whose name is EmployeeType. The declaration occurs above main or in a header file since it will need to be referenced by all parts of the program. No memory has been allocated for the above type until a variable is declared of type EmployeeType.

**Memory Allocation**

Given the following C++ statements:

struct StudentType {

string firstName;

string lastName;

char courseGrade;

int testScore;

int programmingScore;

double gpa;

};

int main(){

StudentType newStudent,

currentStudent;

The memory allocated for the above two variables, newStudent and currentStudent is as follows:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  | | --- | --- | --- | --- | | newStudent | | | | |  |  |  |  | |  | firstName | |  | |  | lastName | |  | |  | courseGrade | |  | |  | testScore | |  | |  | programmingScore | |  | |  | gpa | |  | | |  |  |  |  | | --- | --- | --- | --- | |  | currentStudent | |  | |  |  |  |  | |  | firstName | |  | |  | lastName | |  | |  | courseGrade | |  | |  | testScore | |  | |  | programmingScore | |  | |  | gpa | |  | |

Note: newStudent and currentStudent each have unique addresses just like if you declared two variables of the same datatype. For example int x, y;

**Dot Operator**

To access the members of the struct datatype you utilize the dot operator (.). The syntax is:

**structVarName.member\_name**

You can use either any C++ statement to access the member names or field names of the structVarName as long as you use the dot operator (.).

Example:

cout << “Enter the first name of the student. “;

cin >> newStudent.firstName;

avgScore = (newStudent.testScore + newStudent.programmingScore)/ 2;

**Relational Operators on structs**

Comparison of structVarNames is done via the member\_name of the variable. The following operation is invalid:

if(newStudent == currentStudent)

Whereas the following is valid:

if (newStudent.lastName == currentStudent.lastName)

**Input/Output**

No aggregate I/O is allowed with structures. All I/O is accomplished by accessing each member of the struct via the structVarName one at a time.

struct Variables and Functions

* A struct variable can be passed by reference or value. Reference is preferred due to memory constraints.
* A function can return a value of type struct.

When a struct variable is passed by value a member-wise copy is performed.

Example

#include <iostream>

#include <string>

using namespace std;

struct StudentType {

string firstName;

string lastName;

char courseGrade;

int testScore;

int programmingScore;

double gpa;

};

void readStudentInfo(StudentType &s);

void assignStudentGrade(StudentType &s);

void displayStudentInfo(const StudentType &s); // simulate call by value

int main(){

StudentType newStudent,

currentStudent;

readStudentInfo(currentStudent);

assignStudentGrade(currentStudent);

displayStudentInfo(currentStudent);

return 0;

}// end main

void readStudentInfo(StudentType &s){

cin >> s.firstName >> s.lastName >> s.programmingScore

>> s.testScore >> s.gpa;

}// end readStudentInfo

void assignStudentGrade(StudentType &s) {

int score = (s.programmingScore + s.testScore) / 2;

if (score >= 90)

s.courseGrade = 'A';

else if (score >= 80)

s.courseGrade = 'B';

else if (score >= 70)

s.courseGrade = 'C';

else if (score >= 60)

s.courseGrade = 'D';

else

s.courseGrade = 'F';

}// end assignStudentGrade

void displayStudentInfo(const StudentType &s) {

cout << s.firstName << " " << s.lastName << endl

<< s.courseGrade

<< s.testScore

<< s.programmingScore

<< s.gpa << endl << endl;

}// end displayStudentInfo

**Arrays in structs**

A member of struct can also be an array. For example:

const int SIZE = 100;

struct ListType{

int listElement[SIZE];

int length;

}; // end ListType

The above defines a member of a struct that is an integer array. To access each element of the array you must use the dot operator with the subscript variable.

For example,

ListType intList;

intList.listElement[0] = 56;

intList.listElement[1] = 76;

for(int i = 2; i < intList.length ; i++)

intList.listElement[i] = 3 \* i + 1;

**Arrays of structs**

Given the following struct declaration

struct EmployeeType{

string firstName;

string lastName;

string address1;

string address2

string empId;

double salary;

};

then in main you can declare an array whose elements of type EmployeeType as follows:

EmployeeType employees[10];

This allocates an array of 10 locations where each location is of EmployeeType.

|  |  |  |  |
| --- | --- | --- | --- |
| 0 |  |  | firstName  lastName  address1  address2  empId  salary |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |

Each array element will contain the above.

To access any member of the structure the dot operator must be used and the correct element of the array must be referenced with a subscript. For example,

cout << employees[1].salary << endl;

**structs within a struct**

Since a structure is just another datatype, a member of another structure can be of that type. Given:

struct EmployeeType{

string firstName;

string lastName;

string address1;

string address2

string empId;

double salary;

};

It is possible to define other structures as follows:

struct EmployeeName{

string firstName;

string lastName;

};

struct EmployeeAddress{

string address1;

string address2;

};

struct Employee{

EmployeeName name;

EmployeeAddress address;

string empId;

double salary;

};

To access the members of EmployeeName through the struct Employee you must use the dot operator as follows.

Employee employee;

employee.name.firstName = “Barney”;

employee.name.lastName = “Rubble”;

**Pointers to structs**

Given the following struct definition

struct StudentType {

string firstName;

string lastName;

char courseGrade;

int testScore;

int programmingScore;

double gpa;

};

StudentType student,

\*ptr;

It is possible to assign the address of the variable student to the ptr variable as follows:

ptr = &student; // pointer to a struct variable

Since the dot operator has a higher precedence then the dereferencing operator, we can access the members of the struct has follows:

(\*ptr).gpa = 3.5;

The arrow operator is normally used instead of the above notation. The arrow operator is formed by the hyphen and the greater than sign as follows: ->

(\*ptr).gpa = 3.5; can be rewritten as ptr -> gpa = 3.5 since they both mean the same thing.

**Lab Exercises**

Given the following statements:

struct NameType {

string first;

string last;

};

struct DateType{

int month;

int day;

int year;

};

struct PersonalInfoType{

NameType name;

int identification;

DateType dob;

};

struct EmployeeType{

NameType name;

int rating;

int identification;

string dept;

double salary;

};

PersonalInfoType person;

PersonalInfoType classList[100];

NameType student;

EmployeeType employees[100];

EmployeeType newEmployee;

Answer the following questions based upon the definitions above.

1. Specify if the statement is valid or invalid. If the statement is invalid, why.

1. person.name.first = “Barney”;

VALID

1. cout << person.name << endl;

INVALID, because there’s no ostream to output “person.name”

1. classList[1] = person;

VALID

1. classList[20].identification = 000011100;

VALID

1. person = classList[20];

VALID

1. student = person.name;

VALID

1. classList.dob.day = 1;

INVALID, classList is an array and no memory slot was specified.

2. Given the above declarations write C++ statements to do the following:

1. Store the following information in newEmployee

name: Mickey Doe

identification: 1111111111

rating: 2

dept: Accounting

salary: 34567.78

newEmployee.name.first = "Mickey";

newEmployee.name.last = "Doe";

newEmployee.rating = 2;

newEmployee.identification = 1111111111;

newEmployee.dept = "Accounting";

newEmployee.salary = 34567.78;

1. Initialize rating to zero in the employees array.

for (int i = 0; i < 100; i++)

{

employees[i].rating = { 0 };

}

1. Copy the information of the 20th employee in the array employees into newEmployee.

employees[20] = newEmployee;

3. On page 138 in your textbook do problem 2.

Peter the postman became bored one night and, to break the monotony of the night shift, he carried out the following experiment with a row of mailboxes in the post office. These mailboxes were numbered 1 through 150, and beginning with mailbox 2, he opened the doors of all the even-numbered mailboxes, leaving the others closed. Next, beginning with mailbox 3, he went to every third mail box, opening its door if it were closed, and closing it if it were open. Then he repeated this procedure with every fourth mailbox, then every fifth mailbox, and so on. When he finished, he was surprised at the distribution of closed mailboxes. Write a program to determine which mailboxes these were.

Capture the console window for each test run and paste it in your word document. Hand in a print-out of your program.

4. Do Lab 7.1 Vectors --- This is a self-guided lab for you to learn all the operations of Vectors.

**What to turn in:**

1. Printouts of your word document and your source code.
2. A hard copy of the questions and answers from your lab manual.
3. Compress the word document and the source code from all of your programs and submit them to the D2L assignment folder called Lab Experience Five.

Problem 2 – HEADER

#include <iostream>

using namespace std;

enum mailbox { open, close }; //enum for open/close

const int NUM\_BOX = 150; //number of mailboxes

//prototypes

void closeMailbox(int \*box);

void openMailbox(int \*box);

void countMailbox(int \*box);

Problem 2 – Main

#include "Postman.h"

int box[NUM\_BOX];

int main()

{

closeMailbox(box);

openMailbox(box);

countMailbox(box);

return 0;

}

Problem 2 – Implementation

#include <iostream>

#include "Postman.h"

using namespace std;

//Closes all mailboxes

//Precons: Box array

//Postcons: None

void closeMailbox(int \*box)

{

int i;

for (i = 1; i <= NUM\_BOX; i++) //loop to close all boxes

{

box[i] = close;

}

}

//Opens every even number mailbox first

//Then recurs with every 3rd, 4th, 5th and so on.

//Precons: Box array

//Postcons: None

void openMailbox(int \*box)

{

int e;

for (e = 2; e <= NUM\_BOX; (e += 2)) //open every other box starting with two

{

box[e] = open;

}

for (int k = 3; k <= NUM\_BOX; k++)

{

for (int q = k; q<NUM\_BOX; q += k)

{

if (box[q] == close)

box[q] = open;

else // box must be open

box[q] = close;

}

}

}

//Counts the number of total closed mailboxes

//Precons: Box array

//Postcons: None

void countMailbox(int \*box)

{

int j;

for (j = 1; j <= NUM\_BOX; j++)

{

if (box[j] == close)

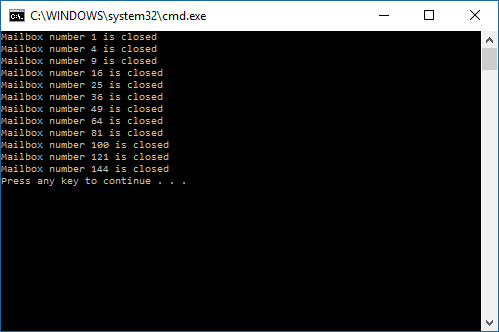
{

cout << "Mailbox number " << j << " is closed" << endl;

}

}

}



Lab 7.1 – vectorlab.cpp

#include <iostream>

#include <vector>

using namespace std;

template <typename T>

ostream & operator<<(ostream & out, const vector<T> & v)

/\*-------------------------------------------------------------------------

Overloaded output operator for vector<T>s.

Precondition: ostream out is open.

Postcondition: Elements of v have been output to out and out is returned.

--------------------------------------------------------------------------\*/

{

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

out << v[i] << " ";

return out;

}

int main()

{

// Declare 6 vectors v1, v2, v3, v4, v5, v6 to illustrate the

// various types of declarations (and constructors)

vector<int> v1;

vector<int> v2(2);

int numInts;

cout << "Enter capacity of v3: ";

cin >> numInts;

vector<int> v3(numInts);

vector<int> v4(3, 99);

// The preceding declaration should work, but it may not in some

// versions of some compilers. The following is a work-around:

// vector<int> v4(3);

// for (int i = 0; i < 3; i++) v4[i] = 99;

//--- End of work-around

int a[] = { 1, 4, 9, 16, 25 };

vector<int> v5(a, a + 5);

vector<int> v6;

//--- 1 --- Add:

// Statements to display the capacity and size of each vector<int>

// and whether it is empty

cout << "1 lab" << endl;

cout << v1.capacity() << ' ' << v1.size() << ' ' << v1.empty() << endl;

cout << v2.capacity() << ' ' << v2.size() << ' ' << v2.empty() << endl;

cout << v3.capacity() << ' ' << v3.size() << ' ' << v3.empty() << endl;

cout << v4.capacity() << ' ' << v4.size() << ' ' << v4.empty() << endl;

cout << v5.capacity() << ' ' << v5.size() << ' ' << v5.empty() << endl;

cout << v6.capacity() << ' ' << v6.size() << ' ' << v6.empty() << endl;

//--- 2 --- Add:

// Statements to display the maximum capacity of a vector<int>

cout << "2 lab" << endl;

cout << v1.max\_size() << endl;

//--- 3 --- Add:

// Statements to see the effect of the reserve() member function

cout << "3 lab" << endl;

v4.reserve(7);

cout << v4.capacity() << ' ' << v4.size() << ' ' << v4.empty() << endl;

//--- 4 --- Add:

// Output statements of the form cout << vector-variable << endl;

// to display the contents of each vector

cout << "4 lab" << endl;

cout << v1 << endl;

cout << v2 << endl;

cout << v3 << endl;

cout << v4 << endl;

cout << v5 << endl;

cout << v6 << endl;

//--- 5 --- Add:

// Statements to append 11 to v2 and then output v2's size and contents

// append 22 to v2 and then output v2's size and contents

// append 33 to v2 and then output v2's size and contents

// remove the last element of v2 and then output v2's size

// and contents

cout << "5 lab" << endl;

v2.push\_back(11);

cout << v2.size() << ' ' << v2 << endl;

v2.push\_back(22);

cout << v2.size() << ' ' << v2 << endl;

v2.push\_back(33);

cout << v2.size() << ' ' << v2 << endl;

v2.pop\_back();

cout << v2.size() << ' ' << v2 << endl;

//--- 6 --- Statements to investigate how capacities grow

// Add statements to append 111 to v1 and then output v1's capacity, size,

// and contents

cout << "6 lab" << endl;

v1.push\_back(111);

cout << v1.capacity() << ' ' << v1.size() << ' ' << v1 << endl;

//--- 7 --- Statements to investigate how capacities grow

// Add statements to append 222, 333, 444, and 555 to v1 and output

// v1's capacity, size, and contents after each value is appended

cout << "7 lab" << endl;

v1.push\_back(222);

cout << v1.capacity() << ' ' << v1.size() << ' ' << v1 << endl;

v1.push\_back(333);

cout << v1.capacity() << ' ' << v1.size() << ' ' << v1 << endl;

v1.push\_back(444);

cout << v1.capacity() << ' ' << v1.size() << ' ' << v1 << endl;

v1.push\_back(555);

cout << v1.capacity() << ' ' << v1.size() << ' ' << v1 << endl;

//--- 8 --- Statements to investigate how capacities grow

// Remove the comment delimiters from the following:

cout << "8 lab" << endl;

int oldCapacity = v1.capacity();

for (int i = v1.size() + 1; i <= 2500; i++)

{

v1.push\_back(999);

if (v1.capacity() == v1.size())

cout << "\n\*\*\* v1 is full with " << v1.size() << " elements\n";

if (v1.capacity() > oldCapacity)

{

cout << "Adding an element increases capacity from "

<< oldCapacity << " to " << v1.capacity() << endl;

oldCapacity = v1.capacity();

}

}

//--- 9 --- Statements to see if element type affects how capacities grow

// Add:

// A declaration of an empty vector<double> v0;

// A loop like the preceding but with v1 replaced by v0

//

// Then change double to char and run it again.

cout << "9 lab" << endl;

vector<char> v0;

for (int i = v0.size() + 1; i <= 2500; i++)

{

v0.push\_back(999);

if (v0.capacity() == v0.size())

cout << "\n\*\*\* v0 is full with " << v0.size() << " elements\n";

if (v0.capacity() > oldCapacity)

{

cout << "Adding an element increases capacity from "

<< oldCapacity << " to " << v0.capacity() << endl;

oldCapacity = v0.capacity();

}

}

//--- 10 --- Statements to see how initial capacity affects

// how capacities grow

// Uncomment the following line:

cout << "10 lab" << endl;

cout << "Initial capacity of v4 is " << v4.capacity() << endl;

// Add a loop like that in 9 but output changes in v4's capacity

for (int i = v4.size() + 1; i <= 2500; i++)

{

v4.push\_back(999);

if (v4.capacity() == v4.size())

cout << "\n\*\*\* v4 is full with " << v4.size() << " elements\n";

if (v4.capacity() > oldCapacity)

{

cout << "Adding an element increases capacity from "

<< oldCapacity << " to " << v4.capacity() << endl;

oldCapacity = v4.capacity();

}

}

//--- 11 --- Statements to access the ends of a vector

// Uncomment the following line:

cout << "11 lab" << endl;

cout << "Original contents of v5: " << v5 << endl;

// Add statements to:

// Output the first and last elements of v5

// Change the first element to 77 and the last element to 88

// Output the contents of v5

v5.front() = 77;

v5.back() = 88;

cout << "Altered contents of v5: " << v5 << endl;

//--- 12 --- Statements to demonstrate correct and incorrect

// use of the subscript operator

// Add statements that try using the subscript operator to:

// change the value in location 1 of v2 to 2222

// append the value 3333 to v2

// append a value to empty vector v6

cout << "12 lab" << endl;

v2[1] = 2222;

cout << v2 << endl;

/\*v2[v2.size()] = 3333;

for (int i = 0; i <= v2.size(); i++)

cout << v2[i] << " ";

cout << endl;\*/

cout << v2.capacity() << ' ' << v2.size() << ' ' << v2.empty() << endl;

//--- 13 --- Add statements that:

// assign v5 to v3

// check if they are equal

// check if v5 is less than v2

// swap contents of v5 and v2

// check if v5 is less than v2

cout << "13 lab" << endl;

v5 = v3;

v5 == v3;

if (v5 == v3)

{

cout << "true" << endl;

}

else

cout << "false" << endl;

if (v5<v2)

{

cout << "true" << endl;

}

else

cout << "false" << endl;

v2 == v5;

if (v5<v2)

{

cout << "true" << endl;

}

else

cout << "false" << endl;

}

