Programming with C++

COMP2011: Structure — a Collection of Heterogeneous Objects

Cecia Chan
Cindy Li
Brian Mak

Department of Computer Science & Engineering
The Hong Kong University of Science and Technology
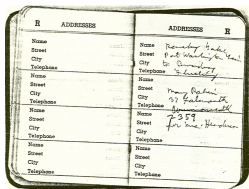Hong Kong SAR, China

Part I

C++ Structure

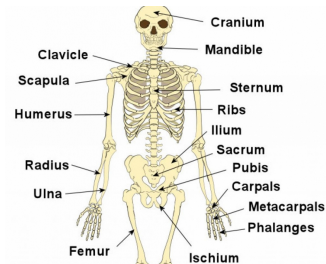| Name	Age	Score
Adam	20	55.6
Bob	18	90.3
Calvin	19	88.0
Dominic	22	76.8
Eddie	30	99.9
Fred	25	47.1

record

array

What is a structure?

- A **structure** is, in general, a collection of **heterogeneous** objects — different kinds of objects. (c.f. array which is a collection of **homogeneous** objects.)

(a) address book

(b) human body

- It is equivalent to **record** in Pascal.

- C++ allows you to define a **new user-defined** data type using the keyword "**struct**".

C++ struct

- Each object in a struct is called its member.
- The data types of various members of a struct can be the same or different.
- The member types can be basic data type, user-defined data type, or a pointer to the new struct currently being defined!
- The struct definition just defines a new user-defined data type, not an object. It is usually defined globally.

```
/* File: point.h */

struct Point
{
    double x;
    double y;
};
```

Syntax: . Operator to Access a struct Member

<struct-variable>.<member-variable>

Examples

```cpp
Point a, b; // a, b contain garbage

// Initialize a Point struct by memberwise assignments
a.x = 24.5;
a.y = 123.0;

// Input/output of a Point struct one member at a time
cin >> a.x >> a.y;
cout << '(' << b.x << ", " << b.y << ')';
```

- struct-struct assignment is done by memberwise copy: each member of the struct on the RHS is copied to the corresponding member of the same kind of struct on the LHS.
- Even a member array can be copied!

Example

```cpp
struct Example
{
    int x;
    float y[5];
};

// Memberwise copy between 2 structs
Example a, b;
b = a;

// Similar to but different from memberwise assignments
b.x = a.x;
b.y = a.y; // Error: arrays can't be assigned to each other!!!
```

Initialization of a struct Variable

- Just like an array variable, a struct variable can be initialized when it is defined using the initializer list with braces.

```
Point a = { 24.5, 123.0 };
```

- If it is not initialized during its definition, later its members can only be modified using separate memberwise assignments or struct-struct assignment (memberwise copy).

```
b.x = 24.5;     // Separate memberwise assignments
b.y = 123.0;    // if no similar object to copy from

// struct-struct assignment to copy a's members to b's
b = a;
```

- For relatively big structures, write a function to do that.

```
void init_point(Point& p, float x, float y)
{
    p.x = x; p.y = y; // Memberwise initialization
}
```

```cpp
#include <iostream>      /* File: point-test.cpp */
using namespace std;
#include "point.h"
#include "point-distance.cpp"

int main()          /* To find the length of the sides of a triangle */
{
    Point a, b, c;
    cout << "Enter the co-ordinates of point A: "; cin >> a.x >> a.y;
    cout << "Enter the co-ordinates of point B: "; cin >> b.x >> b.y;
    cout << "Enter the co-ordinates of point C: "; cin >> c.x >> c.y;

    cout << endl << "Results: " << endl;
    print_distance(a, b);
    print_distance(b, c);
    print_distance(c, a);
    return 0;
}
```

```cpp
/* File: point-distance.cpp */
#include <cmath>          // Header file for C math lib

/* To find the 2D Euclidean distance between 2 points */
double euclidean_distance(const Point& p1, const Point& p2)
{
    double x_diff = p1.x - p2.x, y_diff = p1.y - p2.y;
    return sqrt(x_diff*x_diff + y_diff*y_diff);
}

void print_point(const Point& p)
{
    cout << '(' << p.x << ", " << p.y << ')';
}

void print_distance(const Point& p1, const Point& p2)
{
    cout << "Distance between "; print_point(p1);
    cout << " and "; print_point(p2);
    cout << " is " << euclidean_distance(p1, p2) << endl;
}
```

```cpp
enum Dept { CSE, ECE, MATH }; /* File: student-record.h */

struct Date
{
    unsigned int year;
    unsigned int month;
    unsigned int day;
};

struct Student_Record
{
    char name[32];
    unsigned int id;
    char gender;
    Dept dept;
    Date entry;
};

// Global constants for department names
const char dept_name[][30]
  = {"Computer Science", "Electrical Engineering", "Mathematics"};
```

Access Members of the Student Record struct

```cpp
#include <cstring> // Load the lib header file for strcpy

Student_Record x;  // x contains garbage
strcpy(x.name, "Adam");
x.id = 12345;
x.gender = 'M';
x.dept = CSE;

// Notice how members of nested structures can be assigned
// values through successive use of the dot operator
x.entry.year = 2006;
x.entry.month = 9;
x.entry.day = 1;
```

- Initialize using the braces while it is defined.

```
Student_Record a = { "Adam", 12345, 'M', CSE, { 2006, 9, 1 } };
```

- Initialize using a function:

```
void init_date(Date& x,
    unsigned int year,
    unsigned int month,
    unsigned int day)
{
    x.year = year;
    x.month = month;
    x.day = day;
}
```

```
void init_student_record(Student_Record& a,
    const char name[], unsigned int id,
    char gender, Dept dept, const Date& date)
{
    strcpy(a.name, name);

    a.id = id;
    a.gender = gender;
    a.dept = dept;
    a.entry = date; // struct-struct assignment
}
```

Example: Student Record — student-record.cpp

```cpp
#include <iostream>      /* File: student-record.cpp */
#include "student-record.h"
using namespace std;

void print_date(const Date& date) {
    cout << date.year << '/' << date.month << '/' << date.day << endl;
}

void print_student_record(const Student_Record& x) {
    cout.width(12); cout << "name: " << x.name << endl;
    cout.width(12); cout << "id: " << x.id << endl;
    cout.width(12); cout << "gender: " << x.gender << endl;
    cout.width(12); cout << "dept: " << dept_name[x.dept] << endl;
    cout.width(12); cout << "entry date: "; print_date(x.entry);
}

int main()
{
    Student_Record a = { "Adam", 12345, 'M', CSE, { 2006, 9, 1 } };
    print_student_record(a); return 0;
}
```

Part II

Array of Structures

Array of Structures

You may create an *array* of basic data types as well as user-defined data types, such as *structures*.

student_record sr[3];

"Adam"
12000
'M'
CSE

"Bob"
11000
'M'
MATH

"Cathy"
10000
'F'
ECE

(The above figure ignores the Date member of the Student Record.)

```cpp
#include <iostream>        /* File: student-record-array.cpp */
using namespace std;
#include "student-record-functions.cpp"

int main()
{
    Student_Record sr[] = {
        { "Adam",   12000, 'M', CSE,  { 2006, 1, 10 } },
        { "Bob",    11000, 'M', MATH, { 2005, 9, 1  } },
        { "Cathy",  10000, 'F', ECE,  { 2006, 8, 20 } }
    };

    for (int j = 0; j < sizeof(sr)/sizeof(Student_Record); ++j)
        print_student_record(sr[j]);

    return 0;
}
```

```cpp
/* File: student-record-functions.cpp */
#include <cstring>
#include "student-record.h"

void print_date(const Date& date)
{
    cout << date.year << '/' << date.month << '/' << date.day << endl;
}

void print_student_record(const Student_Record& x)
{
    cout.width(12); cout << "name: " << x.name << endl;
    cout.width(12); cout << "id: " << x.id << endl;
    cout.width(12); cout << "gender: " << x.gender << endl;
    cout.width(12); cout << "dept: " << dept_name[x.dept] << endl;
    cout.width(12); cout << "entry date: "; print_date(x.entry);
}
```

```cpp
void init_date(Date& x, unsigned int year,
               unsigned int month, unsigned int day)
{
    x.year = year;
    x.month = month;
    x.day = day;
}

void init_student_record(Student_Record& a, const char name[],
                         unsigned int id, char gender,
                         Dept dept, const Date& date)
{
    strcpy(a.name, name);
    a.id = id;
    a.gender = gender;
    a.dept = dept;
    a.entry = date;      // struct-struct assignment
}
```

```cpp
#include <iostream>       /* File: sort-student-record.cpp */
using namespace std;
#include "student-record-functions.cpp"

void swap_SR(Student_Record& x, Student_Record& y) {
    Student_Record temp = x; x = y; y = temp;
}

void sort_3SR_by_id(Student_Record sr[]) {
    cout << "#records = " << sizeof(sr)/sizeof(Student_Record) << endl; // ???
    if (sr[0].id > sr[1].id) swap_SR(sr[0], sr[1]);
    if (sr[0].id > sr[2].id) swap_SR(sr[0], sr[2]);
    if (sr[1].id > sr[2].id) swap_SR(sr[1], sr[2]);
}

int main() {
    Student_Record sr[] = {
        { "Adam",  12000, 'M', CSE,  { 2006, 1, 10 } },
        { "Bob",   11000, 'M', MATH, { 2005, 9, 16 } },
        { "Cathy", 10000, 'F', ECE,  { 2006, 8, 27 } } };
    sort_3SR_by_id(sr);
    for (int j = 0; j < sizeof(sr)/sizeof(Student_Record); j++)
        print_student_record(sr[j]);
    return 0;
}
```

Example: Return by Reference Again

```cpp
#include <iostream>        /* File: student-record-rbv.cpp */
using namespace std;
#include "student-record-functions.cpp"

Student_Record& smaller_id(Student_Record& x, Student_Record& y)
{
    return (x.id < y.id) ? x : y;
}

int main() /* To allow student will smaller ID to transfer to CSE */
{
    Student_Record a = { "Amy", 12000, 'F', MATH, { 2006, 1, 10 } };
    Student_Record b = { "Bob", 11000, 'M', MATH, { 2005, 9, 21 } };

    cout << "<<< Before changing department >>>" << endl;
    print_student_record(a); print_student_record(b);
    smaller_id(a, b).dept = CSE;

    cout << "\n\n<<< After changing department >>>" << endl;
    print_student_record(a); print_student_record(b);
    return 0;
}
```

Question: What happens if smaller_id() returns by value?