C++ Structures
Programming Workshop 2 (CSCI 1061U)

Faisal Qureshi
http://faculty.uoit.ca/quareshi
University of Ontario Institute of Technology

C++ struct

struct keyword can be used to define new data types in C++.

Defining a struct

The following code defines a Rect data type that can be used to store the height and the width of a rectangle.

```cpp
struct Rect {
    int h;
    int w;
};
```

We can also do the following

```cpp
typedef struct {
    int h;
    int w;
} Rect;
```

This tells the compiler that we are defining a struct named Rect. This struct contains 2 variables inside of it: an int named h and an int named w. These variables that are part of the struct are called members (or fields). Currently no memory is allocated for this struct. This simply tells the compiler how the struct Rect looks like.

Declaring variables

To use this struct, we can declare a variable of type Rect

```cpp
Rect r1;
```

Above statement declares a variable of type Rect. It allocates the memory necessary to hold struct contents. It is possible to see how much memory a struct takes by using the sizeof() function. E.g., `sizeof(Rect)` will return the size (in bytes) of this struct.

Accessing members

We use the . (dot) operator to access the members as follows

```cpp
r1.h = 23;
r1.w = 90;
```
The above lines set the height and the width of \texttt{r1} equal to 23 and 90, respectively.

As with normal variables, struct member variables are not initialized, and will typically contain junk. We must initialize them manually.

**Assignment**

Assignment works as expected.

\begin{verbatim}
Rect r1;
r1.h = 23;
r1.w = 90;

Rect r2 = r1;
\end{verbatim}

\texttt{r2} is a copy of \texttt{r1}, i.e., it too represents a rectangle of height 23 and width 90.

**Reading and writing to binary files**

The following code writes and read struct from a file.

\begin{verbatim}
Rect r1;
r1.h = 23;
r1.w = 90;

ofstream f("tmp.dat", ios::binary);
f.write((char*) &r1, sizeof(Rect));
f.close();

and for reading
Rect r;
ifstream f("tmp.dat", ios::binary);
f.read((char*) r1, sizeof(Rect));
f.close();
\end{verbatim}

The following, however, won’t work.

\begin{verbatim}
Rect r1;
r1.h = 23;
r1.w = 90;

ofstream f("tmp.txt");
f << r1;
f.close();

or
Rect r;
ifstream f("tmp.txt");
f >> r;
f.close();
\end{verbatim}

This is because struct is a user-defined type. And currently operators >> and << are not defined for it.

**Pointers to struct**

Similar to regular variables, it is possible to declare a pointer to a struct variable.
Rect r;
...

Rect* pR;

pR = &r;
Notice that we use & to get the address of the struct variable (similar to other variables).

**Accessing members when using struct pointers**

Use the -> operator to access the members as follows:

Rect r;
Rect* pR;
pR = &r;
pR->h = 3;
pR->w = 34;

Now the variable r represents a rectangle with height 3 and width 34.

**Initializing struct**

Let’s consider the following struct that represents an Employee

```cpp
struct Employee {
    long id;
    short age;
    double wage;
};
```

It is possible to define and initialize a variable of type Employee in one go as follows:

Employee e = {1033242, 34, 100000.34};

cout << e.id << endl;  // prints 1033242
cout << e.age << endl;  // prints 34
cout << e.wage << endl;  // prints 100000.34

**Using struct with functions**

It is possible to pass the entire struct to a function.

```cpp
#include <iostream>
using namespace std;

struct Rect {
    int h;
    int w;
};

int area(Rect r)
{
    return r.h * r.w;
}

int main()
```
```cpp
{  
Rect r = {5, 23};

cout << "Area of rectangle = " << area(r) << endl;

    return 0;
}
**It is also possible to return a struct from a function.* This is one of the few ways to return multiple values from a function.

#include <iostream>
using namespace std;

struct Employee {
    long id;
    short age;
    double wage;
};

Employee make_CEO(int id, int age) {
    Employee e;
    e.id = id;
    e.age = age;
    e.wage = 10000.0;
    return e;
}

int main()
{
    Employee e = make_CEO(1, 24);

    return 0;
}

Nest struct

struct can contain other struct

struct Employee
{
    short id;
    int age;
    double wage;
};

struct Company
{
    Employee CEO; // Employee is a struct within the Company struct
    int numberOfEmployees;
};

Company c;
Use the . (dot) operator to get to the members

c.CEO.age = 25;
```
This sets the age of the CEO of company c equal to 25.

**Struct alignment**

Typically, the size of a struct is the sum of the size of all its members, but not always!

The size of the struct below is 8 as expected: \((2*\text{sizeof(int)})\).

```c
struct Rect {
    int h;
    int w;
};
```

The size of the struct below on my machine is 12, which is odd since char only takes 1 byte.

```c
struct Rect {
    char a;
    int h;
    int w;
};
```

In general, the size of a struct is at least as large as the size of all the variables it contains. *But it could be larger.* For performance reasons, the compiler will sometimes add gaps into structures (this is called padding).