COMSC-165 Lecture Topic 8
Intro To Object Oriented Programming

Reference

Tutorial 1
Tutorial 2
Tutorial 3

C/C++ `struct` Syntax

group items of different data types
or unrelated data
struct "specifications" (or "definitions")

```
struct tod // the "specification"
{
  int hour; // a "data member"
  int minute; // a "data member"
  int seconds; // a "data member"
}; // note the SEMICOLON
struct myStruct{int a, b, c;};
```

may be local or global

`struct`-based Objects

declarations of struct variables (or "objects")

```
struct tod{...} noon, midnight;
tod noon, midnight;
```

reference `noon.hour`

Using `struct`s And Objects

uses

```
a = b + noon.hour;
noon.hour++;
if (noon.hour == 0)
cout << noon.hour;
abs(noon.hour)
```

declaration and initialization

```
tod noon = {12, 0, 0};
noon.hour = 12;
```

arrays as data members

```
int testScores[10];
char name[64];
  strcpy(me.name, "Robert");
```

structs as data members
define `struct address{...};`
define `struct employee{...} you;`
data member `address home;`

```
  strcpy(you.home.city, "Concord");
```

arrays of struct-based objects

`struct`s And Functions

stand-alone "utility" functions (basic)
have `struct`s as parameters (setNoon)
...or as a return type (getNoon)

```
struct tod
{
  int hour;
  int minute;
  int seconds;
};

void printTod(tod&);

int main()
{
  tod noon = {12};
  printTod(noon);
}

void printTod(tod& t)
{
  cout << t.hour <<...
}
```

encapsulated "member" functions (advanced)
written *inside* the `struct`

```
struct tod
{
  int hour;
  int minute;
  int seconds;
  void printTod()
  {
    cout << hour <<...
  }
};

int main()
{
  tod noon = {12};
  noon.printTod();
}
```

`class`es And `struct`s

`class` and `struct` keywords are *interchangeable*
difference: "private" vs "public" members
by convention, use `struct` when
all members are "public"

```
student students[35];
students[0].score[0] = 99;
```

using struct-based objects in functions
  as a return type
  as an input in the parameter list
  by reference in the parameter list

```
void getNoon(tod& t);
```

using the `const` keyword

Limitations

`struct`s cannot have self-instances
  as data members
but they can have "pointers"

`sizeof`

consistent result for a given struct
may include "padding"

```
struct X  // 8 bytes
{
  char a; // 1 byte
  char b; // 1 byte
  int c; // 4 bytes
};
```

```
struct time
{
  ...
};

class date
{
  public:
  ...
  private:
  ...
};
```

Source Code Organization

`struct` (or `class`) **.h** and **.cpp** files
including in a multi-file project
move `struct` definition and function protos
  to a "header file" (tod.h)
move function definitions
  to a separate CPP file (tod.cpp)
attach to main CPP with a #include:

```
#include "tod.h"
```

a tod-based example...