COMSW 1003-1

Introduction to Computer Programming in C

Lecture 10  Spring 2011

Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html
Announcements

Change in Office Hours this week

1 hour Wednesday, Feb 23rd, 12pm-1pm
1 hour Saturday, Feb 26th, 11am-12pm
Today

• Preprocessor (from Lecture 9)

• Advanced C Types
Advanced Types - Struct

- Arrays group variables of the **same** type
- Structs group variables of **different** types

Struct definition

```c
struct structName {
    fieldType fieldNameval1;
    fieldType fieldNameval2;
    ...
    fieldType fieldNamevalN;
};
```

Once we define the struct, we can use `structName` as if were a type, to create variables!
**Advanced Types - Struct**

**Example:** we want to build a database with the name, age and grade of the students in the class

<table>
<thead>
<tr>
<th>Student 1</th>
<th>Student 2</th>
<th>Student N</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name:</td>
<td>Name:</td>
<td>Name:</td>
</tr>
<tr>
<td>Age:</td>
<td>Age:</td>
<td>Grade:</td>
</tr>
<tr>
<td>Grade:</td>
<td>Grade:</td>
<td>Grade:</td>
</tr>
</tbody>
</table>

```
struct student {
    char name[100];
    int age;
    double grade;
};

struct student st1;
```

*st1 is a variable of type struct!*
In order to access struct fields, we need to use the . operator

```c
struct student {
    char    name[100];
    int     age;
    double  grade;
};

struct student st1, st2;

st1.age = 3;
st2.age = st1.age - 10;
```

`st1.age` is a variable of type `int`, I can use it as a regular variable!
Advanced Types - Struct

We can initialize a struct variable at declaration time, just like with arrays

```c
struct student {
    char name[100];
    int age;
    double grade;
};

struct student st1 = {"mike", 22, 77.4};
```

The initialization fields must be consistent with the fields types!
**Advanced types - Typedef**

**typedef** is used to define a new type

```
typedef type nameOfNewType;
```

```c
typedef int myInt;
myInt c = 3;
```

C is of type **myInt**, which is equivalent to **int**

```c
typedef int myIntArray[7];
myIntArray arr;
```

arr is of type **myIntArray**, which is equivalent to an array of **7 int**

```c
for(c=0; c<7; c++){
    arr[c] = 1;
}
```
Advanced types - Typedef

typedef is used to define a new type

```c
typedef struct student {
    char name[100];
    int age;
    double grade;
};

struct student stud;
stud st1, st2;

st1.age = 3;
st2.age = st1.age - 10;
```

```c
struct student st1, st2;
st1.age = 3;
st2.age = st1.age - 10;
```
Advanced Types - Union

• Similar to struct, but all fields share same memory
• Same location can be given many different field names

```c
struct value{
    int    iVal;
    float fVal;
};

union value{
    int    iVal;
    float fVal;
};
```

We can use the fields of the union only one at a time!
Advanced Types - Enum

• Designed for variables containing only a limited set of values

• Defines a set of **named integer constants**, starting from 0

```c
enum name{ item1, item2, ... , itemN};

enum dwarf { BASHFUL, DOC, DOPEY, GRUMPY, HAPPY, SLEEPY, SNEEZY};

enum dwarf myDwarf = SLEEPY;

myDwarf = 1 + HAPPY;  // myDwarf = SLEEPY = 5;

int x = GRUMPY + 1;   // x = 4;

printf("dwarf %d\n",BASHFUL);  // ‘dwarf 0’
```
Advanced Types - Const

`const` defines a variable whose value cannot be changed

```c
const double PI = 3.14;
double r = 5, circ;
circ = 2 * PI * r;
PI = 7;
```
Advanced Types - Const

`const` defines a variable whose value cannot be changed

```c
const double PI = 3.14;

double r = 5, circ;

circ = 2 * PI * r;

PI = 7;  // Once it’s initialized, a const variable cannot change value
```
**Advanced Types - Const**

`const` defines a variable whose value cannot be changed.

```c
double computeCirc( const double r, const double PI){
    r++; PI++; // Red x
    return(2 * r * PI);
}
/* main function */
int main(){
    const double PI = 3.14;
    double r = 5, circ, circ2;
    circ = 2 * PI * r;
    circ2 = computeCirc(r, PI);
    return 0;
}
```
Advanced Types - Const

`const` defines a variable whose value cannot be changed

declare const double PI = 3.14;
double r = 5, circ, circ2;
circ = 2 * PI * r;
circ2 = computeCirc(r, PI);
return 0;

```c
double computeCirc( double r, const double PI){
    r++;
    PI++;
    return(2 * r * PI);
}
```
Advanced Types - Const

**const** defines a variable whose value cannot be changed

double computeCirc( double r, double PI) {
   r++;
   PI++;
   return (2 * r * PI);
}

/* main function */
int main(){
   **const** double PI = 3.14;
   double r = 5, circ, circ2;
   circ = 2 * PI * r;
   circ2 = computeCirc(r, PI);
   return 0;
}