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

students.c

Advanced Types - Struct

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

Struct definition

```
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

Student 1	Student 2	. . .	Student N
Name:	Name:		Name:
Age:	Age:		Age:
Grade:	Grade:		Grade:

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

struct student st1;
```

st1 is a variable of type struct!

Advanced Types - Struct

In order to access struct fields, we need to use the . operator

```
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

```
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 !

char int double

Advanced types - Typedef

typedef is used to define a new type

```
typedef type nameOfNewType;
```

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

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

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

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

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

Advanced types - Typedef

typedef is used to define a new type

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

struct student st1, st2;

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

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

typedef struct student stud;
stud st1, st2;

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

unions.c

Advanced Types - Union

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

```
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

```
enum name{ item1, item2, … , itemN};
```

```
                0       1     2       3       4       5       6
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'
```

circ.c

Advanced Types - Const

`const` defines a variable whose value cannot be changed

```
const double PI = 3.14;

double r = 5, circ;

circ = 2 * PI * r;

PI = 7;
```

circ.c

Advanced Types - Const

`const` defines a variable whose value cannot be changed

```
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

circ.c

Advanced Types - Const

`const` defines a variable whose value cannot be changed

```
double computeCirc( const double r, const 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;

}
```

circ.c

Advanced Types - Const

`const` defines a variable whose value cannot be changed

```
double computeCirc( double r, const 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;

}
```

circ.c

Advanced Types - Const

`const` defines a variable whose value cannot be changed

```
double computeCirc( double r, double PI){

   r++;   V

   PI++;  V

   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;

}
```