

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

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

Struct definition

Once we define the struct, we can use **structName** as if were a type, to create variables!

4

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!

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

struct student {

st1.age is a variable of type int, I can use it as a regular variable ! char name[100];
int age;
double grade;
};

struct student st1, st2;

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

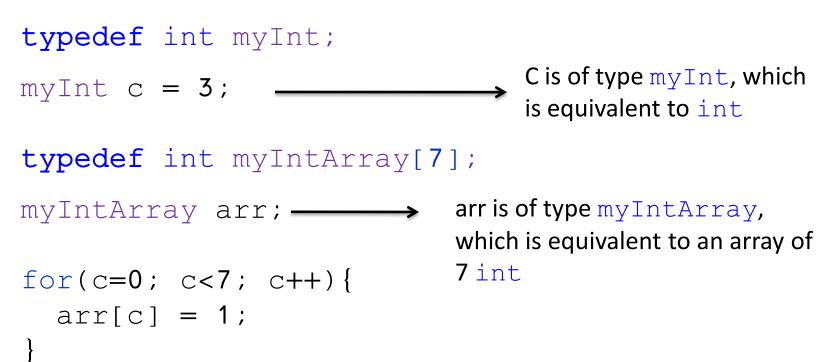
We can initialize a struct variable at declaration time, just like with arrays

```
struct student {
    char name[100];
    int age;
    double grade;
};
struct student st1 = {"mike", 22, 77.4};
```

Advanced types - Typedef

typedef is used to define a new type

typedef type nameOfNewType;



Advanced types - Typedef

typedef is used to define a new type

```
struct student {
struct student {
                               char name[100];
   char name[100];
                               int age;
   int age;
                               double grade;
   double grade;
                            };
};
struct student st1, st2;
                            typedef struct student stud;
                            stud st1, st2;
st1.age = 3;
                            st1.age = 3;
st2.age = st1.age - 10;
                            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

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

iVal

fVal

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

iVal / 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 double PI = 3.14;
double r = 5, circ;
circ = 2 * PI * r;
PI = 7;
```



const defines a variable whose value cannot be changed

```
const double PI = 3.14;
```

```
double r = 5, circ;
```

circ = 2 * PI * r;



Once it's initialized, a const variable cannot change value

```
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;
```

```
double computeCirc( double r, const double PI){
   r++; V
  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;
```

```
double computeCirc( double r, double PI){
  r++; V
  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;
```