NWEN 241
User Defined Types

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington

This Lecture

- Introduction to user defined data types
  - Renaming types
  - Type casting
  - Enumeration types
  - Structure basics

Data Types

- Basic types: int, char, float, double, void, etc
- Derived types: arrays of basic types, pointers to basic types, and functions returning basic types
- Wouldn't it be nice to build user-defined types.

Renaming Types

- **typedef** declares a new name for a specified type
  `typedef type newname;`
  - For example:

```
typedef int Time;    /* Time is an alias of int */
Time hours, minutes, seconds;
```

  - **typedef** does not define a new type
  - A pointer to a function that returns a pointer to a function that returns a pointer to a char

Renaming Types

- **typedef** declares a new name for a specified type

```
typedef type newname;
```

  - For example:

```
typedef int Time;    /* Time is an alias of int */
Time hours, minutes, seconds;
```

  - **typedef** does not define a new type
  - A pointer to a function that returns a pointer to a function that returns a pointer to a char

```
/* char *(*(*)())() */
typedef char *(*(*pfpfpc)())();

pfpfpc a;

/* Or */
```


Renaming Types

- **typedef** declares a new name for a specified type

```
typedef type newname;
```

  - For example:

```
typedef int Time;    /* Time is an alias of int */
Time hours, minutes, seconds;
```

  - **typedef** does not define a new type
  - A pointer to a function that returns a pointer to a function that returns a pointer to a char

```
/* char *(*(*)())() */
typedef char *pc;
typedef pc fpc();
typedef fpc *pfpc;
typedef pfpc fpfpc();
typedef fpfpc *pfpfpc;

pfpfpc a;
```


Type Casting

- We talked about this before
  - Force one variable of one type to be another type

```
(typename)expression;
int i, ii = 5;
float f = 3.14, ff;
i = f;      /* can you do this in java? */
ff = ii;    /* can you do this in java? */
```


Type Casting

- We talked about this before
  - Force one variable of one type to be another type

```
(typename)expression;
int i, ii = 5;
float f = 3.14, ff;
i = (int)f;      /* i=3, f=3.14 or 3.0? */
ff = (float)ii;  /* ff = 5.0 */
```

Type Casting

- We talked about this before
  - Force one variable of one type to be another type

```
(typename)expression;
int i, ii = 5;
float f = 3.14, ff;
i = (int)f;       /* i=3, f=3.14 or 3.0? */
ff = (float)ii;   /* ff = 5.0 */

/* C will do this kind of type casting */
/* automatically, but there are */
/* many cases we have to do */
/* type casting ourselves */
```


Type Casting

- We talked about this before
  - Explicit type casting

```
a = b;  /* if you know a's type, */
        /* but not sure about b's type */;
```


Type Casting

- We talked about this before
  - Explicit type casting

```
a = b;  /* if you know a's type, */
        /* but not sure about b's type */;

a = (a's type)b;
```


Type Casting

- We talked about this before
  - Explicit type casting

```
char week[7][10] = {"Mon", "Tue", ...};
char (*ptrw)[10];
ptrw = week; /* points to the first row */
ptrw++;      /* points to the second row */

/* ptrw – week = 1 */

/* (int)ptrw – (int)week = 10 */
```

Enumeration Types

- A simple example of user-defined types
- Enumerated types contain a list of names

```
enum tag {enumerator list};
```

– For example:

```
enum Colour {Red, Green, Blue, Black} flag;
```

– Use typedef to rename enum Colour

```
typedef enum Colour Colour;
Colour aflag = Red;


Colour suit = Black;
```


Enumeration Types

- A simple example of user-defined types
- Enumerated types contain a list of names

```
enum tag {enumerator list};
```

– For example:

```
enum Colour {Red, Green, Blue, Black} flag;
  /* flag is of type enum Colour */
```

– Use typedef to rename enum Colour

```
typedef enum Colour Colour;
Colour aflag = Red;


Colour suit = Black;
```


Enumeration Types

- A simple example of user-defined types
- Enumerated types contain a list of names

```
enum tag {enumerator list};
```

– For example:

```
enum Colour {Red, Green, Blue, Black} flag;
  /* flag is of type enum Colour */
```

– Use typedef to rename enum Colour

```
typedef enum Colour Colour;
Colour aflag = Red;
   /* declare aflag is of type Colour */
   /* and initialise aflag with Red */
Colour suit = Black;
```


Enumeration Types

- What is behind these names

```
enum Colour {Red, Green, Blue};
```

Enumeration Types

- What is behind these names – integer constants

```
enum Colour {Red, Green, Blue};
```

is automatically defined as:

```
enum Colour {Red=0, Green=1, Blue=2};
```


Enumeration Types

- What is behind these names – integer constants

```
enum Colour {Red, Green, Blue};
```

is automatically defined as:

```
enum Colour {Red=0, Green=1, Blue=2};
```

However, we can override the default values

```
enum Colour {Red=10, Green, Blue};
```

```
enum Colour {Red=3, Green=1, Blue=5};
enum Colour {Red=0, Green=0, Blue=0,
  Yellow=3,...};
```


Enumeration Types

- What is behind these names – integer constants

```
enum Colour {Red, Green, Blue};
```

is automatically defined as:

```
enum Colour {Red=0, Green=1, Blue=2};
```

However, we can override the default values

```
enum Colour {Red=10, Green, Blue};
   /* Green is automatically assigned 11 */
   /* Blue is automatically assigned 12 */
enum Colour {Red=3, Green=1, Blue=5};
enum Colour {Red=0, Green=0, Blue=0,
  Yellow=3,...};
```


Enumeration Types

- Make a Boolean type yourself
- enum vs. #define
  - Both provide a way to associate integer constants with names
  - enum can generate values automatically
- Be aware…

Enumeration Types

- Make a Boolean type yourself
- enum vs. #define
  - Both provide a way to associate integer constants with names
  - enum can generate values automatically
- Be aware…
  - Names used in an enumeration cannot be used in another enumeration within the same scope
  - Names must be valid identifiers

Enumeration Types

- Make a Boolean type yourself
- enum vs. #define
  - Both provide a way to associate integer constants with names
  - enum can generate values automatically
- Be aware…
  - Names used in an enumeration cannot be used in another enumeration within the same scope

```
enum Colour {Red, Green, Blue, Orange};
enum Fruit {Apple, Grape, Orange, Pear};
```

  - Names must be valid identifiers

Enumeration Types

- Make a Boolean type yourself
- enum vs. #define
  - Both provide a way to associate integer constants with names
  - enum can generate values automatically
- Be aware…
  - Names used in an enumeration cannot be used in another enumeration within the same scope

```
enum Colour {Red, Green, Blue, Orange};
enum Fruit {Apple, Grape, Orange, Pear};
```

  - Names must be valid identifiers

```
enum Grade {E, D, ..., A-, A+};
```


Enumeration Types

- An example – use three primary colours

```
enum Colour {Red=0, Green=0, Blue=0, ..., Purple};
typedef enum Colour Colour;

Colour c_array[]={Red, Purple, Black, Green, Orange};

int i, nc = sizeof(c_array)/sizeof(c_array[0]);

for (i =0; i<nc; i++) { /* one of the three primary */
    if(!c_array[i])     /* colours? */
      ...;
    else
      ...;
}
```

Structures

- Structures vs. arrays
  - Members in an array must be of the same type
  - Members in a struct can be of different types

```
struct tag {member1; . . . member n;};
```

- struct is a simplified version of class
  - A class with only public members and no functions
- A struct template

```
struct Person {
    char *name;        /* name[50]? */
    char gender;
    int age;
}; /* struct needs a ";" as array does */
```


Structures

- Use typedef to rename struct Person

```
struct Person {
    char *name;
    char gender;
    int age;
};
typedef struct Person Person;    /* or */
```


Structures

- Use typedef to rename struct Person

```
struct Person {
    char *name;
    char gender;
    int age;
};
typedef struct Person Person;    /* or */

typedef struct {
    char *name;
    char gender;
    int age;
} Person;
```


Structures

- Use typedef to rename struct Person

```
struct Person {
    char *name;
    char gender;
    int age;
};
typedef struct Person Person;

/* what is this? */
struct {
    char *name;
    char gender;
    int age;
} Person;
```

Structures

- Use typedef to rename struct Person

```
/* what is this? */
struct {
    char *name;
    char gender;
    int age;
} Person;

/* this is bad .... */
/* Person is a variable – you cannot declare more */
/* variables of that type */
```


Structures

- Use typedef to rename struct Person

```
struct Person {
    char *name;
    char gender;
    int age;
};
typedef struct Person Person;    /* or */

typedef struct {
    char *name;
    char gender;
    int age;
} Person;
```


Structures

- Let us declare/create a couple of Person objects

```
Person bob, sue;

bob.name = "Robert Jackson";
bob.gender = 'M';
bob.age = 48;

sue.name = "Suzan Jackson";
sue.gender = 'F';
sue.age = 20;
```


Next Lecture

- More on structures and unions