NWEN 241
User Defined Types

Qiang Fu

School of Engineering and Computer Science
Victoria University of Wellington

This Lecture

- More on structures and unions
  - Nested structures
  - Structure initialisation and assignment
  - Pointers to structures
  - Structures with pointers
  - Passing structures to functions
  - Structure sizes
  - Unions

Structures

- Create 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;
```

Structures

- Nested structures
  - Let us add a new member to Person

```
struct Date {
  int day;
  int month;
  int year;
};
typedef struct Date Date;  /* or */

typedef struct {
  int day;
  int month;
  int year;
} Date;
```

Structures

- Nested structures
  - Let us add a new member to Person

```
typedef struct {
  int day;
  int month;
  int year;
} Date;

struct Person {
  char name[50];
  char gender;
  int age;
  Date birthday;
};
```

Structures

- Nested structures
  - Add sue’s birthday

```
Date abirthday = {27, 7, 1989};
                      /* initialisation */
sue.birthday = abirthday;
                      /* assignment */
```

  - Can we do:

```
sue.birthday = {27, 7, 1989};
```

Structures

- Nested structures
  - Add sue’s birthday

```
Date abirthday = {27, 7, 1989};
                      /* initialisation */
sue.birthday = abirthday;
                      /* assignment */
```

  - Can we do:

```
sue.birthday = {27, 7, 1989};    //wrong!
```

Structures

- Be aware…

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

  – Initialisation

```
Person johnb = {"John B", 'M', 18, {12, 3, 1991}};
Person johnh = {"John H", 'M', 32, {12, 3, 1977}};
```

  – Assignment

```
johnb = johnh;
```

  – Can we do this assignment?

```
johnb = {"John H", 'M', 32, {12, 3, 1977}};
```

Structures

- Be aware…
  – Variables of the same struct type can be assigned by one another

```
struct SWEN201 {
  int year;
  int enrolments;
  char *class_rep;
};
typedef struct SWEN201 SWEN201;

SWEN201 sy09, sy2009 = {2009, 40, "Peter"};

sy09 = sy2009;
```

Structures

- Be aware…
  – How about variables of the similar struct types?

```
struct COMP206 {
  int year;
  int enrolments;
  char *class_rep;
};

typedef struct COMP206 COMP206;

COMP206 cy09, cy2009 = {2009, 60, "John"};

cy09 = sy2009;
sy09 = cy2009;
```

Structures

- Be aware…
  – Variables of the similar struct type cannot

```
struct COMP206 {
  int year;
  int enrolments;
  char *class_rep;
};

typedef struct COMP206 COMP206;

COMP206 cy09, cy2009 = {2009, 60, "John"};

cy09 = sy2009;        /* wrong */
sy09 = cy2009;        /* wrong */
```

Structures

- Be aware…
  - If we insist to mix up SWEN and COMP…

```
typedef struct {     /* no tag name here */
  int year;
  int enrolments;
  char *class_rep;
} COMP206, SWEN201;

COMP206 cy09, cy2009 = {2009, 60, "John"};
SWEN201 sy09, sy2009 = {2009, 40, "Peter"};

cy09 = sy2009;
sy09 = cy2009;
```

Structures

- Be aware…
  - If we insist to mix up SWEN and COMP…

```
typedef struct {     /* no tag name here */
  int year;
  int enrolments;
  char *class_rep;
} COMP206, SWEN201;

COMP206 cy09, cy2009 = {2009, 60, "John"};
SWEN201 sy09, sy2009 = {2009, 40, "Peter"};

cy09 = sy2009;       /* accepted */
sy09 = cy2009;       /* accepted */
```

Structures

- Pointers to structures

```
Person *pjohn = &john;

/* modify john's age */

/* use john directly */


/* use a pointer to john */


/* use a pointer to get john, and then use john */
```

Structures

- Pointers to structures

```
Person *pjohn = &john;

/* modify john's age */

/* use john directly */
john.age = 20;

/* use a pointer to john */


/* use a pointer to get john, and then use john */
```

Structures

- Pointers to structures

```
Person *pjohn = &john;

/* modify john’s age */

/* use john directly */
john.age = 20;

/* use a pointer to john */
pjohn->age = 30;

/* use a pointer to get john, and then use john */
```

Structures

- Pointers to structures

```
Person *pjohn = &john;

/* modify john’s age */

/* use john directly */
john.age = 20;

/* use a pointer to john */
pjohn->age = 30;

/* use a pointer to get john, and then use john */
(*pjohn).age = 40;
```

Structures

- Structures with pointer members

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

Person john = {"John B", &anage, &abirthday};

john.name = "John H";   /* ? */
scanf("%s", john.name); /* John Key? */
```

Structures

- Structures with pointer members

```
typedef struct {
  char *name;                      /* name[50]? */
  int *age;
  Date *birthday;
} Person;

Person john = {"John B", &anage, &abirthday};

john.name = "John H";   /* You are fine */
scanf("%s", john.name); /* You may be in trouble*/
```

Structures

- Structures with pointer members

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

Person john = {"John B", &anage, &abirthday};

*john.age = 32;            /* any thing wrong? */

john.birthday->year = 1977;/* any thing wrong? */
```

Structures

- Structures with pointer members

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

Person john = {"John B", &anage, &abirthday};

*john.age = 32; /* "." is of higher precedence */

john.birthday->year = 1977;/*associativity L to R*/
```

Passing Structures to Functions

- Is a structure passed to a function by value?

Passing Structures to Functions

- When a structure is passed to a function, it is passed by value

- But, we can also pass the address of the structure to the function

Passing Structures to Functions

- An example (call-by-value vs. call-by-address)

```
typedef struct {
  ...
} Person;
Person john = {...};       /* initialisation */
-----------------------
john = update(john);       /* update john’s info */
Person update(Person aname)
{ ...
  return aname;
}
-----------------------
update(&john);             /* update john’s info */
void update(Person *ptr)
{ ...
}
```

Passing Structures to Functions

- An example (call-by-value vs. call-by-address)

```
typedef struct {
  char name[50];
  ...
} Person;
Person john = {“John H”, ...}; /* initialisation */

john = update(john);       /* update john’s info */

Person update(Person p)
{
  printf("Printing the old name: %s\n", p.name);
  printf("Type in a new name:\n");
  scanf(" %[^\n]", p.name);    /* “John B” */
  return p;
}
```

Passing Structures to Functions

- An example (call-by-value vs. call-by-reference)

```
typedef struct {
  char name[50];
  ...
} Person;
Person john = {“John H”, ...}; /* initialisation */

update(&john);             /* update john’s info */

void update(Person *p)
{
  printf("Printing the old name: %s\n", p->name);
  printf("Type in a new name:\n");
  scanf(" %[^\n]", p->name);    /* “John B” */
}
```

Size of Structures

- Tell me the sizes of the two structures

```
typedef struct Size1 {
  char achar;
  char bchar;
  char cchar;
  char dchar;
  char echar;
  struct Size1 *next;
} Size1;

typedef struct Size2 {
  int aint;
  int bint;
  char achar;
} Size2;
```

Size of Structures

- Tell me the sizes of the two structures

```
typedef struct Size1 {
  char achar;
  char bchar;
  char cchar;
  char dchar;
  char echar;
  struct Size1 *next;
} Size1;                  /* Size1 = 12 */

typedef struct Size2 {
  int aint;
  int bint;
  char achar;
} Size2;                  /* Size2 = 12 */
```

Unions

- Unions vs. structures
  - Unions follows the same syntax as structures
  - The members of unions have to share storage (only one member can have storage at a time)

```
struct int_and_float {
  int i;
  float f;
} s_number;

union int_or_float {
  int i;
  float f;
} u_number;
u_number.i = 11;
u_number.f = 99.0;
```

Unions

- Unions vs. structures
  - Unions follows the same syntax as structures
  - The members of unions have to share storage (only one member can have storage at a time)

```
struct int_and_float {
  int i;        /* storage allocated to */
  float f;      /* s_number to accommodate */
} s_number;     /* both i and f */

union int_or_float {
  int i;        /* the storage allocated to */
  float f;      /* u_number can accommodate */
} u_number;     /* the largest number (f) */
u_number.i = 11;   /* no storage for f */
u_number.f = 99.0; /* no storage for i */
```

Unions

- What are unions good for
  - Share the same piece of memory between different types of data
  - Reduce the consumption of memory

Next Week/Lecture

- Dynamic memory allocation
- Dynamic data structures