Week 7

XMUT-NWEN 241 - 2024 T2

Systems Programming

Mohammad Nekooei

School of Engineering and Computer Science

Victoria University of Wellington

Content

- Structures

Structures

Background

- Basic data types
  - int : integer ✓
  - char : character ✓
  - float : floating point number ✓
  - double : double-precision floating point number ✓

- Derived data types
  - Arrays ✓
  - Strings ✓
  - **Structures**

Structures

- A `struct` is a derived data type composed of members that are each basic or derived data types
- A single `struct` would store the data for one object. An array of `struct`s would store the data for several objects
- A `struct` can be defined in several ways, as illustrated in the following examples

Declaring a Structure

- Syntax of the structure type declaration:

```
struct structure_tag {
     type1 member1;
     type2 member2;
     ...
} variable_list;
```

  - *structure_tag* specifies the name of the structure
  - *structure_tag* and *variable_list* are optional
  - If *structure_tag* is not specified, *variable_list* should be specified; otherwise, there is no way to declare variables using the unnamed structure type

Declaring a Structure

- Syntax of the structure type declaration:

```
struct structure_tag {
      type1 member1;
      type2 member2;
      ...
} variable_list;
```

- Structure members can be
  - Basic data types
  - Derived and user-defined types
  - *Pointers to basic, derived and user-defined data types*
  - *Function pointers*

Examples

- struct declaration that only defines a type:

```
struct student_info { // named struct
      char name [20];
      int student_id;
      int age;
}; // does not reserve any space
```

- struct declaration that defines a type **and** reserves storage for variables:

```
struct student_info { // named struct
      char name [20];
      int student_id;
      int age;
} s, t; // reserves space for s and t
```

Examples

- Declaring a variable `struct current_student`

```
struct student_info current_student;
```

- Above statement reserves space for:
  - 20 character array,
  - integer to store student ID, and
  - integer to store age

Examples

- Declaring array of structures to store information of enrolled students in a class

```
struct student_info nwen241class[99];
```

- Reserves space for 99 element array of records (`structs`) for students enrolled in NWEN241.

Creating New User Defined Types

- Instead of writing `struct student_info` every time we declare a variable, we can **define** it as a new data type

```
typedef struct {
      char name [20];
      int student_id;
      int age;
} StudentInfo;
```

- This makes `StudentInfo` a new user-defined type, and you can declare a variable as follows:

```
StudentInfo current_student;
```

New struct and Data Type

- If `struct student_info` has been previously defined, then we can create a new data type using `typedef` :

```
typedef struct student_info StudentInfo;
```

Initializing at Declaration (1)

- It is possible to initialize a `struct` at declaration

```
typedef struct {
      char name [20];
      int student_id;
      int age;
} StudentInfo;

StudentInfo current_student = { "John Doe", 12345, 18 };
```

- Order of initializer values should follow order of declaration

Initializing at Declaration (1)

- Partial initialization is also possible

```
typedef struct {
      char name [20];
      int student_id;
      int age;
} StudentInfo;

StudentInfo current_student = { "John Doe", 12345};
```

- Remaining fields will be set to 0

Initializing at Declaration (2)

- It is possible to initialize certain fields of `struct` using **designated initialization**

```
typedef struct {
        char name [20];
        int student_id;
        int age;
} StudentInfo;

StudentInfo s1 = { .age = 18, .name = "John Doe" };
// or StudentInfo s1 = { age: 18, name: "John Doe" };
```

- Initialization can be in any order

Accessing and Manipulating structs

- We can reference a component of a structure by the **direct component selection operator**, which is a **period**, e.g.

```
strcpy(student1.name, "John Smith");
student1.age = 18;
printf("%s is in age %d\n", student1.name, student1.age);
```

- The **direct component selection operator** has level 1 priority in the operator precedence
- Copying of an entire structure can be easily done by the assignment operator

```
student2 = student1;
```

Example – struct and typedef (1)

```
#include <stdio.h>
#include <string.h>

int main() {

   typedef struct student_info {
       char name[20];
       int student_id;
       int age;
   } StudentInfo;

   StudentInfo current_student;     // declare new variable using
                                    // new type StudentInfo
   struct student_info new_student; // declare using struct format

   // do stuff - see next slide
```

Example – struct and typedef (2)

```
    // declarations in previous slide

    // initialize new student record
    strcpy(new_student.name , "John Smith");
    new_student.student_id = 300300300;
    new_student.age = 22;

    // copy new_student to current_student
    current_student = new_student;

    printf("Student name : %s\n", current_student.name);
    printf("Student ID   : %.9d\n", current_student.student_id);
    printf("Student Age  : %d\n", current_student.age);

}
```

Passing struct to Functions (1)

- Suppose there is a structure defined as follows

```
typedef struct student_info {
       char name[20];
       int student_id;
       int age;
   } StudentInfo;
```

Passing struct to Functions (2)

- When a structure variable is passed as an input argument to a function, all its component values are **copied** into the local structure variable

```
/*
 * Display all components of StudentInfo structure
 */
void print_student(StudentInfo s)
{
   printf("Student name: %s\n", s.name);
   printf("Student ID: %d\n", s.student_id);
   printf("%s  is in age %d\n\n", s.name , s.age);
}
```

Passing `struct` to Functions (3)

- Passing entire copy of a structure can be inefficient, especially for large structs
- There is a better way to pass `structs` to functions using pointers

Structures and Pointers

- A **`struct`** pointer can be used to point to a **`struct`**
- Example:

```
typedef struct {
     char name [20];
     int student_id;
     int age;
} StudentInfo;

StudentInfo s = { "John Doe", 12345, 20};
StudentInfo *sp = &s;
```

Accessing and Manipulating struct Pointers

- We use direct component selection operator: period, e.g.,

```
strcpy((*sp).name, "John Smith");
(*sp).age = 18;
printf("%s is in age %d\n", (*sp).name, (*sp).age);
```

- We can reference a component of a structure pointer by the **indirect component selection operator**, which is a **->**, e.g.

```
strcpy(sp->name, "John Smith");
sp->age = 18;
printf("%s is in age %d\n", sp->name, sp->age);
```

- The **indirect component selection operator** has level 1 priority in the operator precedence

Call by Reference for Efficiency

- Recall: When a structure variable is passed as an input argument to a function, all its component values are **copied** into the local structure variable

- Passing entire copy of a structure can be inefficient, especially for large structs

- **For efficiency, pass a copy of the address of structure to function**

- **This can be done using pointer to `struct` as function parameter**

```
typedef struct student_info {
      char name[40];
      int student_id;
      int age;
} StudentInfo;

void print_student(StudentInfo s)
{
      printf("Name: %s\n", s.name);
      printf("Student ID: %d\n", s.id);
      printf("Age: %d\n", s.age);
}

…

StudentInfo s1 = {"John", 12345, 20};
print_student(s1);
```

Call by Reference for Efficiency

```
typedef struct student_info {
      char name[40];
      int student_id;
      int age;
} StudentInfo;

void print_student(StudentInfo *s)
{
      printf("Name: %s\n", s->name);
      printf("Student ID: %d\n", s->id);
      printf("Age: %d\n", s->age);
}

…

StudentInfo s1 = {"John", 12345, 20};
print_student(&s1);
```

Copy of address of `s1` is passed instead of a copy of the entire structure `s1`

Call by Reference

```
…

void print_student(StudentInfo *s)
{
    printf("Name: %s\n", s->name);
    printf("Student ID: %d\n", s->id);
    printf("Age: %d\n", s->age);

    s->age = 1000;
}


…
```

- `print_student()` can actually modify the value of `s`

Call by Reference: Placing Restrictions

```
…

void print_student(const StudentInfo *s)
{
    printf("Name: %s\n", s->name);
    printf("Student ID: %d\n", s->id);
    printf("Age: %d\n", s->age);

    s->age = 1000; // compiler will not
                   // allow this
}


…
```

- How to restrict function from modifying parameters passed by reference?
- Add `const` modifier to parameter

Call by Reference for Efficiency

```
typedef struct student_info {
      char name[40];
      int student_id;
      int age;
} StudentInfo;

void enter_student(StudentInfo *s)
{
      scanf("%[^\n]s", s->name);
      scanf("%d", &s->student_id);
      scanf("%d", &s->age);
}

…

StudentInfo s1;
enter_student(&s1);
```

- If we define a variable as follows to store data to be read in:
- `StudentInfo s1;`

- For the following function, we call it by passing the parameter by reference:
- `enter_student(&s1);`

Why not &?
name is a string, so it is a pointer value and & is not needed.

Copy of address of `s1` is passed instead of a copy of the entire structure `s1`

Array of Structures

```
typedef struct student_info {
      char name[40];
      int student_id;
      int age;
} StudentInfo;


StudentInfo nwen241[80];

strcpy(nwen241[3].name, “John”);
nwen241[3].student_id = 300922023;
nwen241[3].age = 21;
```

nwen241[0].age

	.name	.student_id	.age
nwen241[0]	“Mo”	300981683	21
nwen241[1]	“Saskia”	300961592	18
nwen241[2]	“Pondy”	300182652	25
nwen241[3]	“Kerese”	300922023	24
…		…	…
nwen241[79]	“Peter”	300139414	22

nwen241[3].student_id

Array of structure to function

```
typedef struct student_info {
      char name[40];
      int student_id;
      int age;
} StudentInfo;

void enter_all_student(StudentInfo *s)
{
   for(int i = 0; i < 80; i++ ){
      scanf("%[^\n]s%*c", s[i].name);
      scanf("%d%*c", &s[i].student_id);
      scanf("%d%*c", &s[i].age);

   }
}
…
StudentInfo nwen241[80];
enter_all_student(nwen241);
```

read and ignore '\n'

```
typedef struct student_info {
      char name[40];
      int student_id;
      int age;
} StudentInfo;

void enter_all_student(StudentInfo *s)
{
   for(int i = 0; i < 80; i++ ){
      scanf("%[^\n]s%*c ", s->name);
      scanf("%d%*c", &s->student_id);
      scanf("%d%*c", &s->age);
      s++;
   }
}
…
StudentInfo nwen241[80];
enter_all_student(nwen241);
```