Lab 02: C programming review - Structure, Memory Allocation, and File I/O

CSCI3150 - Introduction to Operating Systems

Zelin DU
1155187968@link.cuhk.edu.hk

Structure

Defining a Structure

To define a structure, you must use the **struct** statement. The structure tag is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition. At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional.

```
struct [structure tag] {

   member definition;
   member definition;
   ...
   member definition;
} [one or more structure variables];
```

```
struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
} book;
```

Reference: https://www.tutorialspoint.com/cprogramming/c_structures.htm

Accessing Structure Members

To access any member of a structure, we use the member access operator (.).

Results:

```
Book 1 title : C Programming
Book 1 author : Author1
Book 1 subject : C Programming Tutorial
Book 1 book_id : 1
Book 2 title : Golang Programming
Book 2 author : Author2
Book 2 subject : Golang Programming Tutorial
Book 2 book_id : 2
```

Reference: https://www.tutorialspoint.com/cprogramming/c_structures.htm

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

struct Books {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};

int main( ) {

   struct Books Book1;        /* Declare Book1 of type Book */
   struct Books Book2;        /* Declare Book2 of type Book */

   /* book 1 specification */
   strcpy( Book1.title, "C Programming");
   strcpy( Book1.author, "Author1");
   strcpy( Book1.subject, "C Programming Tutorial");
   Book1.book_id = 1;

   /* book 2 specification */
   strcpy( Book2.title, "Golang Programming");
   strcpy( Book2.author, "Author2");
   strcpy( Book2.subject, "Golang Programming Tutorial");
   Book2.book_id = 2;

   /* print Book1 info */
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);

   /* print Book2 info */
   printf( "Book 2 title : %s\n", Book2.title);
   printf( "Book 2 author : %s\n", Book2.author);
   printf( "Book 2 subject : %s\n", Book2.subject);
   printf( "Book 2 book_id : %d\n", Book2.book_id);

   return 0;
}
```

Pointers to Structures

```c
#include <stdio.h>
#include <string.h>
struct Books {
    char title[50];
    char author[50];
    char subject[100];
    int book_id;
};
/* function declaration */
void printBook( struct Books *book );
int main( ) {
    struct Books Book1; /* Declare Book1 of type Book */
    struct Books Book2; /* Declare Book2 of type Book */
    /* book 1 specification */
    strcpy( Book1.title, "C Programming");
    strcpy( Book1.author, "Author1");
    strcpy( Book1.subject, "C Programming Tutorial");
    Book1.book_id = 1;
    /* book 2 specification */
    strcpy( Book2.title, "Golang Programming");
    strcpy( Book2.author, "Author2");
    strcpy( Book2.subject, "Golang Programming Tutorial");
    Book2.book_id = 2;
    /* print Book1 info by passing address of Book1 */
    printBook( &Book1 );
    /* print Book2 info by passing address of Book2 */
    printBook( &Book2 );
    return 0;
}
void printBook( struct Books *book ) {
    printf( "Book title : %s\n", book->title);
    printf( "Book author : %s\n", book->author);
    printf( "Book subject : %s\n", book->subject);
    printf( "Book book_id : %d\n", book->book_id);
}
```

Reference: https://www.tutorialspoint.com/cprogramming/c_structures.htm

Memory Allocation

Functions

The C programming language provides several functions for memory allocation and management. malloc( ) and free( ) are most widely used ones. These functions can be found in the <stdlib.h> header file.
Function and Description

```
void *malloc(int num); This function allocates an array of num bytes and leaves them uninitialized.

void free(void *address); This function releases a block of memory block specified by address.
```

While programming, if you are aware of the size of an array, then it is easy and you can define it as an array. For example, to store the name of any person, it can go up to a maximum of 100 characters, so you can define something as follows –

```
char name[100];
```

Reference: https://www.tutorialspoint.com/cprogramming/c_memory_management.htm

Example

Results:

```
Name = Bob
Description: Bob a CSE student in CUHK
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    char name[100];
    char *description;
    strcpy(name, "Bob");
    /* allocate memory dynamically */
    int size = 200;
    description = (char *)malloc( size * sizeof(char) );
    if( description == NULL ) {
        fprintf(stderr, "Error - unable to allocate required memory\n");
    }
    else {
        strcpy( description, "Bob a CSE student in CUHK");
    }
    printf("Name = %s\n", name );
    printf("Description: %s\n", description );
    free(description);
}
```

Reference: https://www.tutorialspoint.com/cprogramming/c_memory_management.htm

File I/O

Functions

```
int open(const char *pathname, int oflags);
ssize_t read(int fd, void *buf, size_t count);
ssize_t write(int fd, const void *buf, size_t count);
off_t lseek(int fd, off_t offset, int whence);
```

O_RDONLY: Read only.
O_WRONLY: Write only.
O_RDWR: Read and write.
O_APPEND: The file is opened in append mode and the file offset is positioned at the end of the file.
O_CREAT: If the pathname does not exist, create it as a regular file.
O_TRUNC: If the file already exists and is a regular file and the access mode allows writing (i.e., is O_RDWR or O_WRONLY) it will be truncated to length 0.

Reference: http://man7.org/linux/man-pages/

Example

Results:

```
rbuf1: Hello World!
rbuf2: World!
```

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <stdlib.h>
int main(int argc, char * argv[]) {
    char buf[20]="Hello World!";
    char rbuf1[20], rbuf2[20];
    int fd;
    if( (fd = open("myfile", O_CREAT | O_TRUNC | O_RDWR, S_IRUSR | S_IWUSR
)) < 0){
        printf("Error in open()\n");
        exit(-1);
    }
    if( write(fd, buf, 20) < 0){
        printf("Error in write()\n");
        exit(-1);
    }
    if( lseek(fd, 0, SEEK_SET) < 0){
        printf("Error in lseek()\n");
        exit(-1);
    }
    if( read(fd, rbuf1, 20) < 0){
        printf("Error in read()\n");
        exit(-1);
    }
    printf("rbuf1: %s\n", rbuf1);
    if( lseek(fd, 6, SEEK_SET) < 0){
        printf("Error in lseek()\n");
        exit(-1);
    }
    if( read(fd, rbuf2, 20) < 0){
        printf("Error in read()\n");
        exit(-1);
    }
    printf("rbuf2: %s\n", rbuf2);
    close(fd);
    return 0;
}
```

Exercise

In the following exercise, you will need to read one file: input.txt, which contains 10 lines. Each line contains a positive integer number which we guarantee that it is smaller than 1000. You need to read each integer, add it by one, and write the results line by line to another file called output.txt

Under the folder exercise you can find the file exercise.c which shows as in the right Figure. **Please only submit the exercise.c**

Tips:

1. The newline character in Linux is '\n'
2. sscanf: You can use this function to convert char array to int (reference: http://www.cplusplus.com/reference/cstdio/sscanf/).
3. sprintf: You can use this function to convert int to char array (reference: http://www.cplusplus.com/reference/cstdio/sprintf/).

```c
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

/*

Under the same folder, file input.txt.
Each line contains a positive integer number and we guarantee the number is
smaller than 1000.
You need to read the number in each line, add it by one, and write them
line by line to another file called output.txt

NOTE: Please don't hardcode the results in your program because we change
the content in input.txt when grading.

TIP: You can use sscanf to convert char array to int and sprintf to convert
int to char array.

*/

int main(){
      /* Fill your codes. */
}
```