C Programming Basic – week 2

For HEDSPI Project

Lecturers :
Cao Tuan Dung
Le Duc Trung
Dept of Software Engineering
Hanoi University of Technology

Topics

- Structures, dynamic allocation review

- Memory Image based File operations
- Exercises

Dynamic Allocation

- Array variables have **fixed** size, used to store a fixed and known amount of variables – **known at the time of compilation**

- This size can't be changed after compilation

- However, we don't always know in advance how much space we would need for an array or a variable

- We would like to be able to **dynamically allocate** memory

The `malloc` function

`void * malloc(unsigned int nbytes);`

- The function `malloc` is used to dynamically allocate `nBytes` in memory
- `malloc` returns a pointer to the allocated area on success, `NULL` on failure
- You should **always** check whether memory was successfully allocated
- Remember to `#include <stdlib.h>`

Example - dynamic_reverse_array

```c
int main(void)
{
    int i, n, *p;

    printf("How many numbers do you want to enter?\n");
    scanf("%d", &n);

    /* Allocate an int array of the proper size */
    p = (int *)malloc(n * sizeof(int));
    if (p == NULL)
    {
        printf("Memory allocation failed!\n");
        return 1;
    }
    /* Get the numbers from the user */
    ...
    /* Display them in reverse */
    ...
    /* Free the allocated space */
    free(p);
    return 0;
}
```

Example -dynamic_reverse_array

```c
int main(void)
{
    . . .
       /* Get the numbers from the user */
       printf("Please enter numbers now:\n");
       for (i = 0; i < n; i++)
            scanf("%d", &p[i]);


       /* Display them in reverse */
       printf("The numbers in reverse order are - \n");
       for (i = n - 1; i >= 0; --i)
            printf("%d ",p[i]);
    printf("\n");
    free(p);
    return 0;
}
```

Why casting?

The casting in

p = (int *)malloc(n*sizeof(int));

is needed because **malloc** returns **void *** :

void * malloc(unsigned int nbytes);

The type (**void ***) specifies a general pointer, which can be cast to any pointer type.

Free the allocated memory

`void free(void *ptr);`

- We use **`free(p)`** to free the allocated memory pointed to by **`p`**

- If **`p`** doesn't point to an area allocated by **`malloc`**, a run-time error occurs

- Always remember to free the allocated memory once you don't need it anymore

Exercise 2.1

- Implement the function `my_strcat` :
  - Input – two strings, `s1` and `s2`
  - Output – a pointer to a dynamically allocated concatenation
  - For example: The concatenation of "hello_" and "world!" is the string "hello_world!"

- Test your function

Solution: functionmy_strcat

```c
char *my_strcat(char *str1, char *str2)
{
    int len1, len2;
    char *result;

    len1 = strlen(str1);
    len2 = strlen(str2);

    result = (char*)malloc((len1 + len2 + 1) *
  sizeof(char));
    if (result == NULL) {
        printf("Allocation failed! Check memory\n");
        return NULL;
    }

    strcpy(result, str1);
    strcpy(result + len1, str2);

    return result;
}
```

Solution: main()

```c
int main(void)
{
   char str1[MAX_LEN + 1], str2[MAX_LEN + 1];
   char *cat_str;

   printf("Please enter two strings\n");

   scanf("%100s", str1);
   scanf("%100s", str2);

   cat_str = my_strcat(str1, str2);
   if (cat_str == NULL)
   {
      printf("Problem allocating memory!n");
      return 1;
   }

   printf("The concatenation of %s and %s is %s\n", str1, str2,
   cat_str);
   free(cat_str);

   return 0;
}
```

Structures - User Defined Types

- A collection of variables under a single name.

- A convenient way of grouping several pieces of related information together.

- Variables in a **struct** (short for structure) are called members or fields.

Defining a **struct**

```
struct struct-name
{
    field-type1 field-name1;
    field-type2 field-name2;
    field-type3 field-name3;

    ...

};
```

Example – complex numbers

```
struct complex {
      int real;
      int img;
 };
struct complex num1, num2,
  num3;
```

Typedef

- We can combine the **typdef** with the structure definition:

```
typedef struct complex {
    int real;
    int img;
} complex_t;



complex_t num1, num2;
```

Exercise 2.2

- Given two following structure:

```
typedef struct point
{
    double x;
    double y;
} point_t;


typedef struct circle
{
    point_t center;
    double  radius;
} circle_t;
```

- Write a function is_in_circle which returns 1 if a point p is covered by circle c. Test this function by a program.

Solution

```c
int is_in_circle(point_t *p, circle_t *c)
{
    double x_dist, y_dist;

    x_dist = p->x - c->center.x;
    y_dist = p->y - c->center.y;

    return (x_dist * x_dist + y_dist * y_dist <= c->radius * c->radius);
}

int main(void)
{
    point_t  p;
    circle_t c;

    printf("Enter point coordinates\n");
    scanf("%lf%lf", &p.x, &p.y);
    printf("Enter circle center coordinates\n");
    scanf("%lf%lf", &c.center.x, &c.center.y);
    printf("Enter circle radius\n");
    scanf("%lf", &c.radius);

    if (is_in_circle(&p, &c))
        printf("point is in circle\n");
    else
        printf("point is out of circle\n");

    return 0;
```

Pointers in Structures

- If a member in a **struct** is a pointer, all that gets copied is the *pointer (the address)* itself
- Exercise: Give this type of Student

Working mode for binary file

mode	Description
"rb"	opens an existing binary file for reading.
"wb"	creates a binary file for writing.
"ab"	opens an existing binary file for appending.
"r+b"	opens an existing binary file for reading or writing.
"w+b"	creates a binary file for reading or writing.
"a+b"	opens or create an existing binary file for appending.

File handle: Working with a bloc of data

- Two I/O functions: fread() and fwrite(), that can be used to perform block I/O operations.

- As other file handle function, they work with the file pointer.

fread()

- The syntax for the fread() function is

size_t fread(void *ptr, size_t size, size_t n, FILE *stream);

- ptr is a pointer to an array in which the data is stored.
- size: size of each array element.
- n: number of elements to read.
- stream: file pointer that is associated with the opened file for reading.
- The fread() function returns the number of elements actually read.

fwrite()

- The syntax for the fwrite() function is

size_t fwrite(const void *ptr, size_t size, size_t n, FILE *stream);

- ptr is a pointer to an array that contains the data to be written to an opened file
- n: number of elements to write.
- stream: file pointer that is associated with the opened file for writing.
- The fwrite() function returns the number of elements actually written.

function feof

- int feof(FILE *stream);

- return 0 if the end of the file has not been reached; otherwise, it returns a nonzero integer.

Examples

- Read 80 bytes from a file.

```c
enum {MAX_LEN = 80};
int num;
FILE *fptr2;
char filename2[]= "haiku.txt";
char buff[MAX_LEN + 1];
if ((fptr2 = fopen(filename2, "r")) == NULL){
  printf("Cannot open %s.\n", filename2);
   reval = FAIL; exit(1);
}
. . . .
num = fread(buff, sizeof(char), MAX_LEN, fin);
buff[num * sizeof(char)] = `\0';
printf("%s", buff);
```

Exercise 2.3

- Write a program that use bloc-based file operations to copy the content of lab1.txt to lab1a.txt

- Use: fread, fwrite, feof

Solution

```c
#include <stdio.h>
enum {SUCCESS, FAIL, MAX_LEN = 80};
void BlockReadWrite(FILE *fin, FILE *fout);

main(void) {
    FILE *fptr1, *fptr2;
    char filename1[]= "lab1a.txt";
    char filename2[]= "lab1.txt";
    int reval = SUCCESS;

    if ((fptr1 = fopen(filename1, "w")) == NULL){
        printf("Cannot open %s.\n", filename1);
        reval = FAIL;
    } else if ((fptr2 = fopen(filename2, "r")) == NULL){
    printf("Cannot open %s.\n", filename2);
    reval = FAIL;
    } else {
     BlocReadWrite(fptr2, fptr1);
     fclose(fptr1);
     fclose(fptr2);
    }
    return reval;
}
```

Solution

```
void BlockReadWrite(FILE *fin, FILE *fout) {
    int num;
    char buff[MAX_LEN + 1];

    while (!feof(fin)){
     num = fread(buff, sizeof(char),
         MAX_LEN, fin);
     buff[num * sizeof(char)] = `\0';

     printf("%s", buff);
     fwrite(buff, sizeof(char), num, fout);
    }
}
```

Solution

```c
#include <stdio.h>
enum {SUCCESS, FAIL, MAX_LEN = 80};
void BlockCat(FILE *fin);

main(int argc, char* argv[]) {
    FILE *fptr1, *fptr2;
    int reval = SUCCESS;
    if (argc !=2){
        printf("The correct syntax should be: cat1
                filename \n");
        reval = FAIL;
    }

    if ((fptr1 = fopen(argv[1], "r")) == NULL){
        printf("Cannot open %s.\n", argv[1]);
        reval = FAIL;
    } else {
     BlocCat(fptr1);
     fclose(fptr1);
    }
    return reval;
}
```

Solution

```c
void BlockCat(FILE *fin) {
    int num;
    char buff[MAX_LEN + 1];

    while (!feof(fin)){
     num = fread(buff, sizeof(char),
         MAX_LEN, fin);
     buff[num * sizeof(char)] = `\0';

     printf("%s", buff);
    }
}
```

Exercise 2.4

A)Improve the program in previous exercise so that it accepts the two filenames as command arguments.

For example: if your program is named "filecpy". You can use it as the following syntax (in Linux):
./filecpy haiku.txt haiku2.txt

B. Write a program having the same functionality as cat command in Linux
./cat1 haiku.txt

Hint

- Just use the argc[] and argv[]

```
if(argc<3) { printf("%s <file1> <file2>n",argv[0]); exit(1); }
```

- argv[1] and argv[2] will be the name of source file and destination file.

```
if((fp=fopen(argv[1],"r"))==NULL) {
…
};
if((fp2=fopen(argv[2],"w"))==NULL) {
…
};
```

Exercise 2.5: **Input and Output of structure**

- We assume that you make a mobile phone's address book.
- Define a data structure that can store "name," "telephone number," "e-mail address," and make an array of the structures that can hold at most 100 of the data.
- Input about 10 data to this array.
- Write a program to write the array content using fwrite() into the file for the number of data stored, and read the data into the array again using the fread ( ) function.

Solution

```c
#include <stdio.h>

enum {SUCCESS, FAIL, MAX_ELEMENT = 20};

// the phone book structure
typedef struct phoneaddress_t {
    char name[20];
    char tel[11];
    char email[25];
}phoneaddress;

int main(void)
{
    FILE *fp;
    phoneaddress phonearr[MAX_ELEMENT];
    int i,n, irc; // return code
    int reval = SUCCESS;
```

Solution

```
printf("How many contacts do you want to enter (<20)?");
scanf("%d", &n);
for (i=0; i<n; i++){
  printf("name:"); scanf("%s",phonearr[i].name);
  printf("tel:"); scanf("%s",phonearr[i].tel);
  printf("email:"); scanf("%s",phonearr[i].email);
}
if ((fp = fopen("phonebook.dat","w+b")) == NULL){
   printf("Can not open %s.\n", "phonebook.dat");
   reval = FAIL;

}

// write the entire array into the file
irc = fwrite(phonearr, sizeof(phoneaddress), n, fp);
printf(" fwrite return code = %d\n", irc);
fclose(fp);
```

Solution

```c
//read from this file to array again
if ((fp = fopen("phonebook.dat","rb")) == NULL){
   printf("Can not open %s.\n", "phonebook.dat");
   reval = FAIL;
}
irc = fread(phonearr, sizeof(phoneaddress), n, fp);
printf(" fread return code = %d\n", irc);
for (i=0; i<n; i++){
  printf("%s-",phonearr[i].name);
  printf("%s-",phonearr[i].tel);
  printf("%s\n",phonearr[i].email);
}
fclose(fp);
return reval;
}
```

File Random Accessing

- Two functions: fseek() and ftell()
- fseek(): function to move the file position indicator to the spot you want to access in a file.
- Syntax

  ```
  fseek(FILE *stream, long offset, int whence);
  ```

- Stream is the file pointer associated with an opened file
- Offset indicates the number of bytes from a fixed position
- Keywords: SEEK_SET, SEEK_CUR, and SEEK_END
  - SEEK_SET: from the beginning of the file
  - SEEK_CUR: from the current position
  - SEEK_END: from the end of file

File Random Accessing

- ftell: obtain the value of the current file position indicator

- Syntax:

  long ftell(FILE *stream);

- rewind(): reset the file position indicator and put it at the beginning of a file

- Syntax:

  void rewind(FILE *stream);

Dynamic memory allocation

- Write a program to load a specific portion of the address book data from the file (for example, "3rd data to 6th data" or "2nd data to 3rd data"), modify something on the data, and finally save the data to the file again.

- But, you must allocate necessary minimum memory (the necessary size for "3rd data to 6th data" is four, while two for "1st data to 2nd data") to save the data by the malloc( ) function.

Solution

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

enum {SUCCESS, FAIL, MAX_ELEMENT = 20};

// the phone book structure
typedef struct phoneaddress {
    char name[20];
    char tel[11];
    char email[25];
}phoneaddress;

int main(void)
{
    FILE *fp;
    phoneaddress *phonearr;

    int i,n, irc; // return code
    int reval = SUCCESS;
    printf("Read from 2sd data to 3rd data \n");
```

Solution

```c
if ((fp = fopen("phonebook.dat","r+b")) == NULL){
    printf("Can not open %s.\n", "phonebook.dat");
    reval = FAIL;
}

 // Memory allocation
phonearr =
(phoneaddress *)malloc(2 * sizeof(phoneaddress));
if (phonearr == NULL)
{
    printf("Memory allocation failed!\n");
    return FAIL;
}

if (fseek(fp,1*sizeof(phoneaddress),SEEK_SET) != 0)
{
    printf("Fseek failed!\n");
    return FAIL;
}
irc = fread(phonearr, sizeof(phoneaddress), 2, fp);
```

Solution

```c
 for (i=0; i<2; i++){
   printf("%s-",phonearr[i].name);
   printf("%s-",phonearr[i].tel);
   printf("%s\n",phonearr[i].email);
 }

 // Modifying some data
 strcpy(phonearr[1].name,"Lan Hoa");
 strcpy(phonearr[1].tel,"0923456");
 strcpy(phonearr[1].email,"lovelybuffalo@hut.edu.vn");

 fseek(fp,1*sizeof(phoneaddress),SEEK_SET);
 irc = fwrite(phonearr, sizeof(phoneaddress), 2, fp);
 printf(" fwrite return code = %d\n", irc);
 fclose(fp); free(phonearr);

 return reval;
}
```

Exercise 2.6

- Given a text file call class1EF.txt. Write a program to insert a space line between each line in file's content.

Solution

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

void    double_space(FILE *, FILE *);
void    prn_info(char *);

int main(int argc, char **argv)
{
    FILE    *ifp, *ofp;

    if (argc != 3) {
        prn_info(argv[0]);
        exit(1);
    }
    ifp = fopen(argv[1], "r");      /* open for reading */
    ofp = fopen(argv[2], "w");      /* open for writing */
    double_space(ifp, ofp);
    fclose(ifp);
    fclose(ofp);
    return 0;
}
```

Solution for homework

```c
void double_space(FILE *ifp, FILE *ofp)
{
    int  c;

    while ((c = getc(ifp)) != EOF) {
        putc(c, ofp);
        if (c == '\n')
    putc('\n', ofp);    /* found a newline - duplicate
                      it */

    }
}


void prn_info(char *pgm_name)
{
    printf("\n%s%s%s\n\n%s%s\n\n",
        "Usage:  ", pgm_name, "  infile  outfile",
        "The contents of infile will be double-spaced
",
        "and written to outfile.");
}
```

Formatted Input Output

- Two function fprintf and fscanf
- Work as printf and scanf.

```
int fscanf(FILE *stream, const char *format, …);
int fprintf(FILE *stream, const char *format, …);

fprintf(fp, "%d %s", 5, "bear");
fscanf(fp, "%d %s", &n, s);
```

Homework

- Write a program to read some numbers from the standard input and output them to an "out.txt" file in reverse order. In addition, output a sum of the numbers to the end of out.txt.

- The format loaded from the standard input is that the 1st number is the number of data, and the 2nd and proceeding numbers are for process. In a case when you input

- 4 12 -45 56 3

- "4" is the number of numbers that follows, and the remainder "12 -45 56 3" should be an output numbers onto the "out.txt" file. The output to "out.txt" is,

- 3 56 -45 12 26

- The last number "26" is the sum of four numbers.

- Because the number of numbers you input changes each time, you must dynamically allocate memory for the number of data: using the malloc( ) function.

Solution for homework

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
//author: Cao Tuan Dung
//for HEDSPI project
enum {SUCCESS, FAIL};

int main(void)
{
    FILE *fp;
    int *p;
    int i,n, value, sum;
    int reval = SUCCESS;

    printf("Enter a list of numbers with the first is
    the size of list: \n");
    scanf("%d", &n);
    p = (int *)malloc(n*sizeof(int)); i=0; sum=0;
```

Solution for homework

```c
while(i<n) {
    scanf("%d", &value);
    p[i++]=value;
    sum+=value;
}

if ((fp = fopen("out.txt","w")) == NULL){
    printf("Can not open %s.\n", "out.txt");
    reval = FAIL;
}
for (i=n-1; i>=0;i--){
    fprintf(fp,"%d ",p[i]);
}
fprintf(fp,"%d ",sum);
fclose(fp);
free(p);
return reval;
}
```