# Data structure and algorithms lab

## DYNAMIC MEMORY PROGRAMMING & FILE OPERATION 2

**Lecturers : Cao Tuan Dung**
**email: dungct@soict.hust.edu.vn**
**Dept of Software Engineering**
**Hanoi University of Science and Technology**

## Topics

- Structures, dynamic allocation review
- Memory Image based File operations
- Exercises

2

# 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

3

# 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>`**

4

## Example -dynamic_reverse_array

```
#include <stdlib.h>
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;
}
```

5

## Example -dynamic_reverse_array

```
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;
}
```

6

# 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.

7

# The `calloc` function

```
void * calloc(unsigned int nbytes);
```

- Dynamically allocate the specified number of blocks of memory of the specified type.
- It initializes each block with a default value '0',while malloc() leaves the memory uninitialized.
- Returns a pointer to the allocated area on success, **`NULL`** on failure
- **ptr = (float*) calloc(25,sizeof(float));**

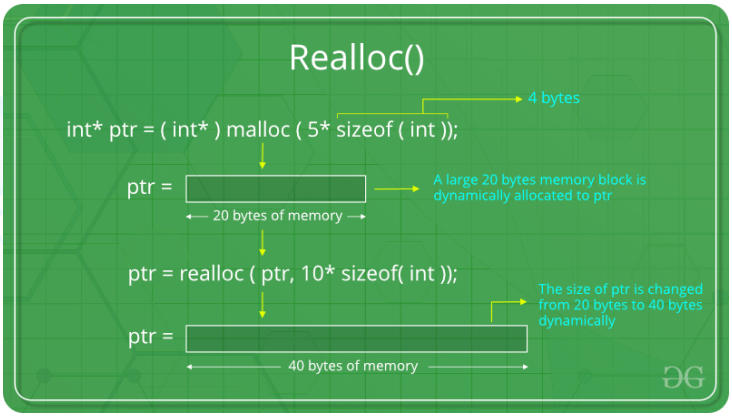
8

# Reallocate Memory

- **void *realloc(void *ptr, size_t size)**
  - resize the memory block pointed to by **ptr** that was previously allocated with a call to **malloc** or **calloc**
- **Parameters**
  - **ptr** -- This is the pointer to a memory block previously allocated with malloc, calloc or realloc to be reallocated.If this is NULL, a new block is allocated and a pointer to it is returned by the function.
  - **size** -- This is the new size for the memory block, in bytes.If it is 0 and ptr points to an existing block of memory, the memory block pointed by ptr is deallocated and a NULL pointer is returned.
- **Return Value**
  - This function returns a pointer to the newly allocated memory, or NULL if the request fails

9

# Illustration of realloc function




10

# Example

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

int main()
{
   char *str;

   /* Initial memory allocation */
   str = (char *) malloc(15);
   strcpy(str, "tutorialspoint");
   printf("String = %s,  Address = %u\n", str, str);

   /* Reallocating memory */
   str = (char *) realloc(str, 25);
   strcat(str, ".com");
   printf("String = %s,  Address = %u\n", str, str);

   free(str);

   return(0);
}
```

```
String = tutorialspoint, Address = 355090448 String = tutorialspoint.com, Address = 355090448
```

11

# 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

12

# Exercise

- 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

13

# Solution: functionmy_strcat

14

## Solution: main()

```
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;
}
```

15

## Homework ICT55

- **Implement and demo the function**
  - **char* subStr(char* s1, int offset, int number)**
- **This function take the subString of s1 that starts from the character with offset index and has number characters.**
- **Remember to check the validation of the parameters. In the case that the number is greater than the number of characters resting from offset to the end of s1. Return the subString of s1 from offset.**

16

# 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.

17

# Defining a `struct`

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

18

# Example – complex numbers

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

19

# Typedef

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

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


complex_t num1, num2;
```

20

# Exercise

- 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.**

21

# Solution

```
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;
```

22

## Pointers in Structures

- If a member in a **`struct`** is a pointer, all that gets copied is the *pointer (the address)* itself



## Homework

- Write a function that checks whether the two circles intersect or not. Use the above function in a program that asks how many circles the user wants to generate and uses dynamic memory to store them. User can choose to:
  - Enter information about each circle manually
  - Automatically generated (random).
- The program prints out:
  - information about the circles
  - the circle intersects with the most different circles (details - which to intersect with)

# Homework

- Make change to the last week exercise program about Class List and Transcript as follows:
- Instead of static arrays, use pointers and dynamically allocate the amount of memory needed for the data stored in the class list file.
- Add « input more » feature: the program asks for the number of additional students and reallocate the memory using realloc function and get input of new students.
- Attention: class list (student list) file must contain at least ten students.

25

# 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. |

26

# 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.

27

# 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.

28

# 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.

29

# function feof

- int feof(FILE *stream);

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

30

# Examples

- Read 80 bytes from a file.

```
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, fptr2);
buff[num * sizeof(char)] = `\0';
printf("%s", buff);
```

31

# Exercise

- Write a program that use bloc-based file operations to copy the content of lab1.txt to to lab1a.txt
- Use: fread, fwrite, feof

32

# Solution

```
#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) {

// Your task is filling the missing code
here….

}
```

# Exercise

- Write program mycat that works like the command cat in Unix
- Using fread funtion.

35

# Solution

```
#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;
}
```

36

## Solution

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

    // fill in the missing code



}
```

37

## Exercise

- 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

38

# Hint

- Just use the argc[] et 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) {
…
};
```

39

# Homework

- Make change to mycat program so that it can display content of big files page by page with the command line parameter –p.

- For example:
  ./mycat  haiku.txt -p

40

# Homework

- Write a multi-modes copy program with menu driven interface providing these functionalities:
  1. Copy by character
  2. Copy by line
  3. Copy by block - optional size

  After finishing copying, print out the execution time on the screen for comparison.
  Attention: source file should be a text file larger than 640KB.

41

# Exercise: 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 named phonebook.dat for the number of data stored, and read the data into the array again using the fread ( ) function.

42

```
enum {SUCCESS, FAIL, MAX_ELEMENT
  = 20};

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

43

## Solution

```
#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;
```

44

# 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);
```

45

# Solution

```
//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;
}
```

46

# Homework (binary file)

- **Continue with last week's exercise on student lists, Write a program that uses the dynamically allocated memory to read the transcript file (bangdiem.txt, the output of last week's exercise) and write it down to the binary file grade.dat (containing the array of student structures).**
- **Then read the file grade.dat and print the transcript on the screen.**
- **Search for a student by student number, update the mark and save to file. Read the file again and display information to the screen to check.**
- **Recommend students to write a program with a menu interface (1. Text2Dat 2. Display Dat file 3. Search and Update. 4 Quit)**

47

# 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
- Whence: 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

48

# 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);

49

# 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.

50

# Convert text data to binary

- Write a program that transforms Vietnamese English dictionary data from text to binary form.
- Ask user the begin and end number then display the words between this range in dict.
- http://www.denisowski.org/Vietnamese/vnedict.txt
- Miên Đất Hứa : the Promised Land Miến : Burma (short for Miến Điện) Miến Điện : Burma Miền Trung : Central Vietnam

51

# Homework: Phone catalog

- Gather information about iPhone, Samsung, Oppo.. smart phones on the web and write to the file PhoneDB.txt in the following format (each line corresponds a phone model, the file needs to contain at least 20 phone models).
  - Model  Memory Space (GB)  Screen Size (inches) Price
- Write a program that has the menu interface as follows:
  1. Import DB from text: automatically converts PhoneDB.txt to PhoneDB.dat using dynamically allocated memory.
  2. Import from DB: Read from file. dat into memory. Support full reading mode and partial reading from the **beginning** of the file or the **end** of the file (the user enters the number of records to read and the starting position).
  3. Print All Database: Read data from PhoneDB.dat to the screen. If there are more than one screen, pagination is required
  4. Search by phone by Phone Model.
  5. Exit

52

# Homework: File split and merge

- **Create phonebook.dat file (output of Lab exercise) containing at least twenty contact entries. Write a filesplit program that takes two parameters: the file name and the number of contacts you want to split, the two file names, and split the original file into two files.**
- **For example with this execution the phone1.dat should includes first 10 contacts and phone2.dat has the rest.**
  - **filesplit phone.dat 10 phone1.dat phone2.dat**
- **Write a merge file program that works with the following syntax:**
  - **filemerge phone1.dat phone2.dat phone.dat**
- **Write a fileread program to read the contents of the .dat files above to check the results of filesplit and filemerge.**

53

# 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);
```

54

# Exercise

- 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.

55

# Solution

```
#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;
```

56

# Solution for homework

```
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;
}
```



# Homework: Header file

- The header is a structure used to manage the general information of a file written in binary form. Write a program whose input is any file (binary, text) - the program automatically inserts the header of the file's file header structure in the form of binary content as follows:
  - File size in bytes (excluding headers)
  - The writer's name
  - The date the header was created.
- Write a program to read the header file on the screen to check the results. If the file does not have a header, the message is "no header".