



Data structure and algorithms lab

WEEK 1

Lecturers : Cao Tuan Dung

**Dept of Software Engineering
Hanoi University of Science and
Technology**



Introduction

- C Programming practice in UNIX environment.
- Programming topics related to [Data Structures and Algorithms]
- Compiler: gcc
- Editor: Emacs, K-Developer.



Evaluation

- Midterm: 30%
 - Evaluated through every week homework. Each student – 2 times during a semester.
 - 3 to 4 students are evaluated at the beginning of the Lab.
- Final: 70%
 - Programming test on computer.
 - The evaluation is not based on source code but on output program.

3



gcc syntax

- Parameter:
 - Wall : turn on all alerts
 - c: make object file
 - o: name of output file
 - g: debug information
 - l: library
- ```
gcc -Wall hello.c -o runhello
./runhello
```



## This week: Basic Data Structures and Algorithms

- Topic:
  - Array, String, Pointer Review
  - Character based File operations in UNIX
  - Programming Exercises



## Array

- A block of many variables of the same type
- Array can be declared for any type
  - E.g. `int A[10]` is an array of 10 integers.
- Examples:
  - list of students' marks
  - series of numbers entered by user
  - vectors
  - matrices


## Arrays in Memory

- Sequence of variables of specified type
- The array variable itself holds the address in memory of beginning of sequence

- Example:

```
double S[10];
```

|     |   |   |   |   |   |   |   |   |   |   |     |
|-----|---|---|---|---|---|---|---|---|---|---|-----|
| ... | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... |
|-----|---|---|---|---|---|---|---|---|---|---|-----|



- The k-th element of array A is specified by A[k-1] **(0-based)**

## Example - reverse

```
#include <stdio.h>

int main(void)
{
 double i, A[10];

 printf("please enter 10 numbers:\n");
 for(i=0; i<10; i++)
 scanf("%f", &A[i]);

 printf("numbers in reversed order:\n");
 for(i=9; i>=0; i--)
 printf("%3.4f\n", A[i]);

 return 0;
}
```



## Exercise: Find a specific element in an array

- Continue the previous program by displaying the element which is closest to the mean value of all array's elements.
- Note: Student must use **CONSTANT** for array **SIZE**.



## Pseudo code for solution

```
avg = average(a, SIZE);
mindiff = 1000000;
for (i=0; i<SIZE; i++){
 diff = fabs(a[i] - avg);
 if (diff < mindiff) {
 mindiff = diff;
 medianindex = i;
 }
}
→ return a[medianindex]
```

## Frequency

- Write a program to input an integer  $n$  ( $n \leq 20$ ) and a list of  $n$  non-negative integer numbers. Output each number and its appearance frequency in the list separated by a space in each line. For example:

| Input     | Output |
|-----------|--------|
| 5         | 1 1    |
| 1 5 4 9 4 | 5 1    |
|           | 4 2    |
|           | 9 1    |

## Exercise

- Write a program that gets an input line from the user (ends with '\n') and displays the number of times each letter appears in it.

The output for the input line: "hello, world!"

The letter 'd' appears 1 time(s).  
The letter 'e' appears 1 time(s).  
The letter 'h' appears 1 time(s).  
The letter 'l' appears 3 time(s).  
The letter 'o' appears 2 time(s).  
The letter 'r' appears 1 time(s).  
The letter 'w' appears 1 time(s).

**Assume all inputs are lower-case!**



## Exercise (20 minutes)

- Implement a function that accepts two integer arrays and returns 1 if they are equal, -1 if they are symmetric, 0 otherwise
- Write a program that accepts two arrays of integers from the user and checks for equality



## Solution

```
#include <stdio.h>

#define SIZE 5

int compare_arrays(int arr1[], int arr2[], int size)
{
 int i = 0;

 for (i = 0; i < size; ++i)
 {
 if (arr1[i] != arr2[i])
 return 0;
 }

 /* if we got here, both arrays are identical */
 return 1;
}
```

## Solution

```
int main(void)
{
 int input1[SIZE], input2[SIZE], i;

 printf("Please enter a list of %d integers:\n",
 SIZE);
 for (i = 0; i < SIZE; ++i) scanf("%d", &input1[i]);

 printf("Please enter another list of %d
 integers:\n", SIZE);
 for (i = 0; i < SIZE; ++i) scanf("%d", &input2[i]);

 if (compare_arrays(input1, input2, SIZE) == 1)
 printf("Both lists are identical!\n");
 else
 printf("The lists are not identical...\n");

 return 0;
}
```

## Home Exercise 1

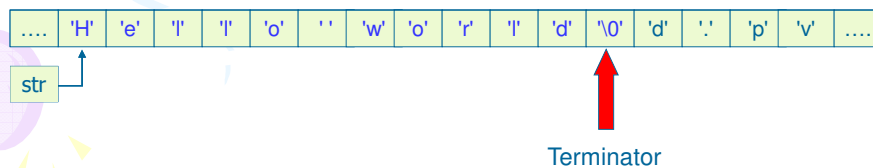
- Redo the program which compares two integer array that not use the size parameter.



# Strings

- An array of characters
- Used to store text
- Another way to initialize:

```
char str[] = "Text";
```



# String

- In order to hold a string of  $N$  characters we need an array of length  $N + 1$
- So the previous initialization is equivalent to

```
char str[] = {'b', 'l', 'a', 'b', 'l',
 'a', '\0'};
```

## String and character related function

- `getchar()`
  - `c = getchar()`
- `scanf`
  - `scanf("%s", str);`
- `gets()`
  - `gets(str);`

## String and character related function

- `strlen(const char s[])`  
returns the length of s
- `strcmp(const char s1[],  
const char s2[])`  
compares s1 with s2
- `strcpy(char s1[],  
const char s2[])`  
copies to contents of s2 to s1
- `strcat(char s1[], char s2[])`  
concatenate to contents of s2 to s1, then store at s1



## Exercise

- write a function that:
  - gets a string and two chars
  - the function scans the string and replaces every occurrence of the first char with the second one.
- write a program to test the above function
  - the program should read a string from the user (no spaces) and two characters, then call the function with the input, and print the result.
- example
  - input: "papa", 'p', 'm'
  - output: "mama"



## Solution



## Exercise: Tokenizer

- Write a program reading a sentence from users. Then display each word in the sentence in a line. A word is defined as a sequence of characters without space. For example:
  - « The house nextdoor is very old. »
  - The
  - house
  - ....



## Exercise

- Write a program that ask user to input a number of students in a class, then their full name in Vietnamese. Print the student list sorted by the first name of student. For example
  - Nguyen Bao Anh
  - Tran Quang Binh
  - Vuong Quoc Binh
  - Dao Thi Ha
  - Ngo Anh Vu
- Print the maximum number of students who have the same first name.



## Homework

- Extend the function in previous exercise by replacing a substring in a string by another string. For example:
- `replace("papa", "pa", "be")` return "bebe".
- Hint: You can use `strstr` to find the substring, ...



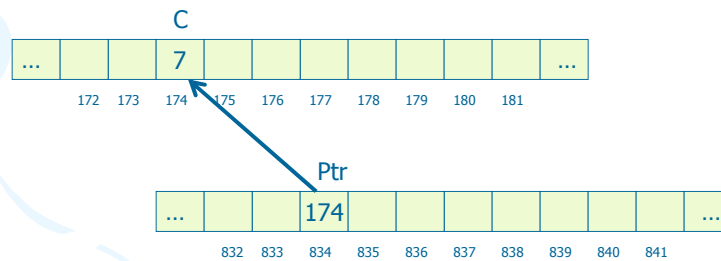
## Pointer - Declaration

```
type *variable_name;
```

- A pointer is declared by adding a \* before the variable name.
- Pointer is a variable that contains an address in memory.
- The address should be the address of a variable or an array that we defined.

# Pointers

- Here ptr is said to *point* to the address of variable c



## Referencing and Dereferencing

```
int n;
int *iptr; /* Declare P as a pointer to int */
n = 7;
iptr = &n;

printf("%d", *iptr); /* Prints out '7' */
*iptr = 177;
printf("%d", n); /* Prints out '177' */
iptr = 177; /* This is unadvisable!! */
```



## Exercises

Write a function that accepts a double parameter and returns its integer and fraction parts.

Write a program that accepts a number from the user and prints out its integer and fraction parts, using this function.

## Solution

```
void split(double num, int *int_part, double *frac_part)
{
 *int_part = (int)num;
 *frac_part = num - *int_part;
}

int main(void)
{
 double num, fraction;
 int integer;

 printf("Please enter a real number: ");
 scanf("%lf", &num);

 split(num, &integer, &fraction);
 printf("The integer part is %d\n", integer);
 printf("The remaining fraction is %lf\n", fraction);

 return 0;
}
```



## Exercise ICT54.2

- Write a program that uses random number generation to create sentences. The program should use four arrays of strings called **article**, **noun**, **verb** and **preposition**. The program should create a sentence by selecting a word at random from each array in the following order: **article**, **noun**, **verb**, **preposition**, **article** and **noun**. As each word is picked, it should be concatenated to the previous words in an array that is large enough to hold the entire sentence. The words should be separated by spaces. When the final sentence is output, it should start with a capital letter and end with a period. The program should generate 10 such sentences.
- The article array should contain the articles "the", "a", "one", "some" and "any";
- the noun array should contain the nouns "boy", "girl", "dog", "town" and "car";
- the verb array should contain the verbs "drove", "jumped", "ran", "walked" and "skipped";
- the preposition array should contain the prepositions "to", "from", "over", "under" and "on".



## Exercise

- Write a function with the prototype:  

```
void replace_char(char *str,
 char c1,
 char c2);
```
- It replaces each appearance of `c1` by `c2` in the string `str`.  
**Do not use the `[]` operator!**
- Demonstrate your function with a program that uses it



## Solution

```
void main(int argc, char* argv[]) {
 if (argc == 1) {
 return;
 }
 int i;
 for (i = 1; i < argc; i++) {
 printf("%s\n", argv[i]);
 }
}
```

## Command line arguments

- Command line arguments are arguments for the **main** function
  - Recall that main is basically a function
  - It can receive arguments like other functions
  - The 'calling function' in this case is the operating system, or another program

## 'main' prototype

```
int main(int argc, char* argv[])
```

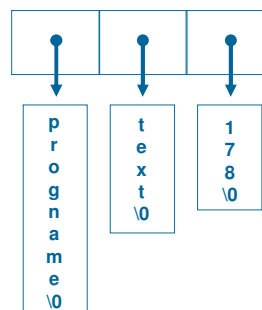
- When we want `main` to accept command line arguments, we must define it like this
  - `argc` holds the number of arguments that were entered by the caller
  - `argv` is an array of pointers to `char` – an array of strings – holding the text values of the arguments
- The first argument is always the program's name

## 'main' prototype

```
int main(int argc, char* argv[])
```

`argc` : 3

`argv` :





## Exercise

- Write a program that accepts two numbers as command line arguments, representing a rectangle's height and width (as floating-point numbers).
- The program should display the rectangle's area and perimeter



## Solution

```
int main(int argc, char* argv[])
{
 double width, height;

 if (argc != 3)
 {
 printf("Wrong number of arguments!\n");
 printf("CORRECT SYNTAX: RECT <WIDTH> <HEIGHT>\n");
 return 1;
 }

 width = atof(argv[1]);
 height = atof(argv[2]);

 printf("The rectangle's area is %f\n", width * height);
 printf("The rectangle's perimeter is %f\n",
 2 * (width + height));

 return 0;
}
```



## Homework: WordReplication

- Write a command line program that replicates a word for n time. For example
- replicate go 7 →gogogogogogogo



## Exercise: Sentence Inverter

- Write a program that take a sentences as a sequence of word in command line and print them in inverse order.



## Homework

- Write a command line program that calculates  $e^x$  with two optional syntax:
  - E 50
  - Or
  - E 50 0.0003



## Home work

- Write a command line program that solves the second degree equation  $ax^2 + bx + c = 0$  with the following syntax: sde a b c
- Example sde 1 2 1 return:  $x_1 = x_2 = -1$



## Homework

- Improve exercise from week 1 that it can take command line arguments.
- replace bachkhoahoabinh oa ii



## File Handling

- **C communicates with files using a new datatype called a file pointer.**
- **File pointer:**
  - references a disk file.
  - used by a stream to conduct the operation of the I/O functions.
- **FILE \*fptr;**



## 4 major operations

- Open the file
- Read from a file → program
- Write to a file: Program → file
- Close the file.



## Opening a file

- `fopen()` function.
- `FILE *fopen(const char *filename, const char *mode);`

```
FILE *fptr;
if ((fptr = fopen("test.txt", "r")) ==
 NULL) {
 printf("Cannot open test.txt file.\n");
 exit(1);
}
```

## Opening a file

- filename: name of the file.
  - It can be a string literal: `"data.txt"`
  - It may contain the full path of the file:  
`"/root/hedspi/CProgrammingBasic/Lab1/data.txt"`
  - It may be a character array that contains the file name:  
`char file_name[] = "junk.txt";`
- **NOTE:** *If the file path is not specified, the file is located in the same folder as the C program.*

## Mode for text file

| mode | Description                                          |
|------|------------------------------------------------------|
| "r"  | opens an existing text file for reading.             |
| "w"  | creates a text file for writing.                     |
| "a"  | opens an existing text file for appending.           |
| "r+" | opens an existing text file for reading or writing.  |
| "w+" | creates a text file for reading or writing.          |
| "a+" | opens or create an existing text file for appending. |






## 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. |



## Closing a file

- The `fclose` command can be used to disconnect a file pointer from a file.
  - `int fclose(FILE *stream);`
- 
- 

## Example: File Open and Close

```
1: /* Opening and closing a file */
2: #include <stdio.h>
3:
4: enum {SUCCESS, FAIL};
5:
6: main(void)
7: {
8: FILE *fptr;
9: char filename[] = "haiku.txt";
10: int reval = SUCCESS;
11:
12: if ((fptr = fopen(filename, "r")) == NULL){
13: printf("Cannot open %s.\n", filename);
14: reval = FAIL;
15: } else {
16: printf("The value of fptr: 0x%p\n", fptr);
17: printf("Ready to close the file.");
18: fclose(fptr);
19: }
20:
21: return reval;
22: }
```

## Reading and Writing Disk Files

- In C, you can perform I/O operations in the following ways:
  - **Read or write one character at a time.**
  - **Read or write one line of text (that is, one character line) at a time.**
  - Read or write one block of characters at a time.



## Character based file operations in UNIX

- **Read or write one character at a time.**

- Character input and output
  - fgetc() and fputc()
- `int fgetc(FILE *stream);`
- `int fputc(int c , FILE *stream);`



## Exercise about file no 1

- Create a text file name lab1.txt with the content as you want.
- Write a program to read from a text file one character at a time, then write it to a new file with the name lab1w.txt



## Solution



## Exercise about file (cont)

- Write a program to read sentences from a specified file one character at a time.
- Each capital letter is converted into a lower-case letter, and each lower-case letter is converted into a capital letter. The new sentence is then written into another file.
- Note that you must output numbers, the signs as they are.



## Homework

- Write the program named mycp that works like command cp. It can copy a text file to another using similar syntax of cp command. For example:  
mycp a1.txt a2.txt

*Attention: Program must check syntax (number of parameters) – print user guide texts..)*



## Homework : Appending

- Write a program that appends the content of file2 to the end of file1. Assume that two files are existing already.
- Syntax:  
- apd <file1> <file2>
- Attention: Check syntax – help user..



## Homework: Convert to Uppercase

- Write a program called `uconvert` that transforms all letters in the content of a given text file to uppercase.
- Syntax: `uconvert tata.txt`
- For example
  - original `tata.txt`: `helloworld`
  - after the program execution : `HELLOWORD.`



## Supplementary homework

- Write a program called `statsfile` that takes a command line parameter that is the name (with or without fullpath) of a text file.
- Then print out the screen statistics of the number of occurrences of the characters (not case-sensitive) included in the file content.
- Run the program with the source program file as input.
- Write the results to the `sourcestats.txt`



## Exercise

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



## Solution for homework

```
#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

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

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

void prn_info(char *pgm_name)
{
 printf("\n%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.");
}
```

## Homework: Caesar cipher

- Write a program that encodes the text file according to the Caesar cipher taking two parameters:
  - file name and the addition offset
- The algorithm's idea is simple as follows: if the addition offset is 3: then the characters will be encoded into the character behind it 3 places in ASCII code. For example  $A \rightarrow D$ ,  $B \rightarrow E$
- Then write a decode program with the same logic.
- B. Improve the program so that the coding is circular in the alphabet. For example  $A \rightarrow D$ , ...,  $Z \rightarrow C$



## Read or write one line at a time.

- Two functions: `fgets()` and `fputs()`
- `char *fgets(char *s, int n, FILE *stream);`
  - `s` references an array that is used to store characters
  - `n` specifies the maximum number of array elements.
- `fgets()` function can read up to `n-1` characters, and can append a null character after the last character fetched, until a newline or an EOF is encountered.

## Read or write one line at a time.

- `int fputs(const char *s, FILE *stream);`
- `s`: array that contains the characters to be written to a file
- return value
  - 0 for success
  - non zero in case of fail.

A decorative graphic in the top-left corner of the slide featuring three balloons: a green one at the top, a blue one in the middle, and a purple one at the bottom. Each balloon has a grid pattern and is surrounded by yellow triangular shapes representing streamers or light rays.

## Exercise

- Redo the exercise F1 but the program will read and write one character line at a time.

A decorative graphic in the top-left corner of the slide featuring three balloons: a green one at the top, a blue one in the middle, and a purple one at the bottom. Each balloon has a grid pattern and is surrounded by yellow triangular shapes representing streamers or light rays.

## Solution



## New Exercise K63: Line counting & cat

- Modify previous program so that it does not copy file but displays the number of lines and the content of source file to the screen.
- Expected User Interface:  
Reading file Haiku.txt.... done!  
Haiku haiku  
Tokyo  
Hanoi  
This file has 3 lines.



## Homework

- Write a program named mycat that read and display on the screen the content of a given file. The command can take 1 or 2 arguments
- cat <filename> : display content to the end
- cat <filename> -p : view page by page.



## Homework

- Write a program whose command line parameter is the path to a text file (size <80 lines). The program adds a new line at the end of the file with the content containing the first characters of the lines in the original file.



## Read and write formatted text

- `int fscanf( FILE *stream, const char *format, ... );`
  - This function works like `scanf` except that it read from a file stream.
- `int fprintf(FILE *stream, const char *format, ... );`
  - The only difference between `fprintf` and `printf` is that `fprintf` can redirect output to a particular stream.



## Exercise

Write a program to read two or more lines into an array of character strings one by one from a specified file and find the length of each line. You must write the length of each line and character string in the file.

For example, one line in an input file

The quick brown fox jumps over the lazy dog.

should be output as follows.

44 The quick brown fox jumps over the lazy dog.



## Solution

*Just Modify the function LineReadWrite using strlen and fprintf*

```
void LineReadWrite(FILE *fin, FILE *fout)
{
 char buff[MAX_LEN];
 int len;
 while (fgets(buff, MAX_LEN, fin) != NULL)
 {
 len = strlen(buff)-1;
 fprintf(fout, "%d %s", len, buff);
 printf("%s", buff);
 }
}
```



## Homework (Lesson 2)

- Create a text file whose content is a class list of at least 6 students. Each line includes the following 4 fields:
  - No Student\_number First-name (do not contain space characters) Phone\_number
    - 1 20181110 Bui\_Van 0903112234
    - 2 20182111 Joshua\_Kim 0912123232
- Then read this information from the file into an array of elements of structural type. Program ask user to input mark (score) for each student and record the results in the file bangdiem.txt (transcript.txt) which has the corresponding score field for each student.



## Exercise on Lab: File merging line by line

- Write a program that takes parameters from the command line
  - merge file1 file2 file3
- The program will read file1 file2 and create file 3 by mixing two files in the following way: after one line from file1, one line from file2 will be copied to output file. Note that the number of lines of file1 file2 may be different. When it is impossible to continue copying alternately line by line, the copy proceeds normally.

## Solution

```
void LineMerge(FILE *f1, FILE *f2, FILE *f3)
{
 char buff1[MAX_LEN], buff2[MAX_LEN];
 int len;
 while ((fgets(buff1, MAX_LEN, f1) != NULL) &&
 (fgets(buff2, MAX_LEN, f2) != NULL)) {
 fputs(buff1, f3);
 fputs(buff2, f3);
 }
 if (buff1 != NULL) fputs(buff1, f3);

 while (fgets(buff1, MAX_LEN, f1) != NULL) {
 fputs(buff1, f3);
 }
 while (fgets(buff2, MAX_LEN, f2) != NULL) {
 fputs(buff2, f3);
 }
}
```

## Source code with line number

- Write a program that convert your source code C to a file that indicate the line number at the beginning of each line of code.



## Homework

- Write a program to read a text file created with emacs. Put a line number to the head of the line and output the contents of the file to the standard output. A text file name can be specified as the argument to the program.
- For example, the following content of a text file  
This is sample file.  
Hello!
- is output as follows.  
1 This is sample file.  
2 Hello!



## Homework: File compare

Write a program to compare two files (at least 10 lines each file) given as the command parameters and indicates:

- the first line where they differ(line numbers).
- all lines where they differ.



A decorative graphic on the left side of the slide featuring three balloons in green, blue, and purple, each with a grid pattern and yellow streamers.

## Homework– Content filter

- Imagine you have to write a program to filter the content printed on the screen. The program replaces all pre-defined sensitive words with strings of special characters (eg \*, #) of the corresponding length. Write a program whose command line parameter is a file and a character that can be used instead. Note white space does not need to be replaced.
- For example: `./filter myfile.txt #` will print to the screen: `#### this weather!`

A decorative graphic on the left side of the slide featuring three balloons in green, blue, and purple, each with a grid pattern and yellow streamers.

See you next time