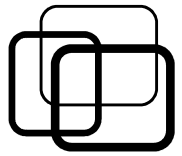
String and Text File

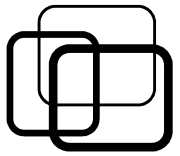
Inst. Nguyễn Minh Huy

Contents



- String.
- Text file.

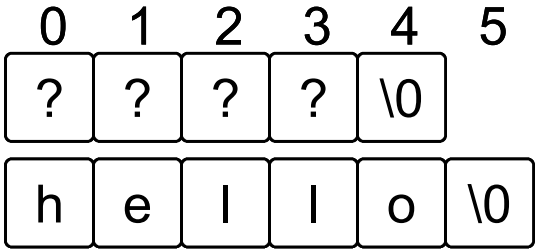
String



- Basic concept:
  - String is an array of chars storing readable text.
  - C string (null-terminated string):
    - Array of chars + '\0' at the end.
    - Declaration: char **<string name>** [<string length> **+ 1**];

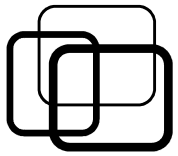
char **s1[5]**;

char **s2[ ] {"hello"}**;



  - C++ string (std::string):
    - Data type in library <string>.
    - Support dynamic string.
    - Declaration: std::string **<string name>**;

String



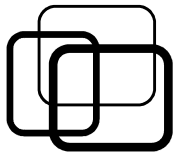
- Basic concept:
  - String comes with helpful libraries:
    - C: <string.h>, <stdio.h>, <ctype.h>.
    - C++: <string>, <iostream>, <sstream>.
  - Iterate string:
    - C string: check for '\0' or use strlen(<string>).

```
for (int i = 0; s[ i ] != '\0'; ++i)
{
        // Process character s[ i ].
}
```

    - std::string: <string>.length().

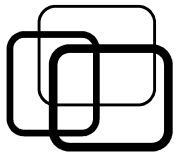
```
for (int i = 0; i < s.length(); ++i)
{
        // Process character s[ i ].
}
```

String



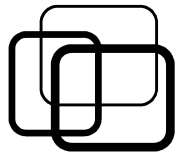
- Input string:
  - C string:
    - fgets(<string>, <max length>, stdin).
    - std::cin.getline(<string>, <max length>) (C++).
  - std::string:
    - std::getline(std::cin, <string>).
- Output string:
  - C string:
    - printf(“%s”, <string>).
    - puts(<string>).
    - std::cout << <string> (C++).
  - std::string: std::cout << <string>.

String



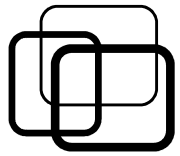
- Practice: <ctype.h>
  - Count words: “the quick fox” → 3 words.
    - Space: ‘ ‘, **‘\t’**, **‘\n’**, **‘\r’** -> **isspace**(<char>).
    - Alphabet: **‘A’–‘Z’**, **‘a’-‘z’** -> **isalpha**(<char>).
    - Punctuation: **‘.‘**, **‘,’**, **‘?’**, **‘!’**,… -> **ispunct**(<char>).
    - Digit: **‘0’**, **‘1’**, **‘2’**, **‘3’**, **‘4’**, … -> **isdigit**(<char>).
  - Capitalize words: “the quick fox” → “The Quick Fox”.
    - Lower alphabet: **‘a’-‘z’** -> **islower**(<char>).
    - Capitalize: <char> - 32 -> **toupper**(<char>).

Contents



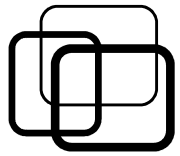
- String.
- **Text file.**

Text file



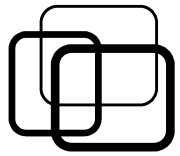
- File IO:
  - Advantages (vs. keyboard and screen):
    - Do not need user:
      - Automatic control..
      - Repeatable.
      - Communicate with other programs.
    - Persistent storage.
  - Disadvantages (vs. RAM):
    - Sequential vs. random access.
    - Slower.

Text file



- File stream:
  - File is either input or output device.
  - File stream is connection between program and file.
  - Declaration:
    - C (library <stdio.h>): **FILE** *<name>.
    - C++ (library <fstream>): **std::fstream** <name>.
  - Steps to process a file:
    - Open file stream.
    - Read/write file stream.
    - Close file stream.

Text file

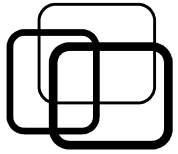


- Open file stream:
  - C syntax:
    - **fopen(**“<file path>”, “<open mode>”**)**;
    - Return: file stream (succeeded) or NULL (error).

      FILE *f = **fopen(**“C:/test.txt”, “**r**”**)**;
      if (**!f**) fprintf(stderr, “Error.”);
      else printf(“Succeeded.”);
  - C++ syntax:
    - **<stream>(**“<file path>”, <C++ open mode>**)**;
    - Check **<stream>** to know the result.

      std::fstream f (“test.txt”, **std::ios::in)**;
      if (**!f**) std::cerr << “Error.”;
      else std::cout << “Succeeded.”;

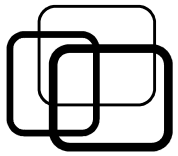
Text file



- Open file stream:
  - C/C++ open modes:

Open mode C/C++	Ý nghĩa
r / std::ios::in	**R**ead-only, open for reading (text mode). Return NULL if file not found.
w / std::ios:out	**W**rite-only, open for writing (text mode). File created or overwritten.
a / std::ios:app	**A**ppend-only, open for append (text mode). File created if not found.
[r/w/a]+ / combine |	Combine read and write (text mode).
[r/w/a]b / combine |	Read and write in binary (binary mode).

Text file

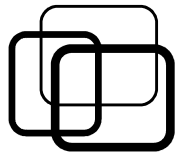


- Close file stream:
  - C syntax:
    - **fclose(**<stream>**)**;
    - Close an opened stream.
    - Always close a stream after used.

```
FILE  *f = fopen(“C:\\test.txt”, “r”);
if ( !f ) {
        fprintf(stderr, “Error.”);
        fclose( f );        // Wrong.
} else {
        printf(“Succeeded.”);
        fclose( f );        // Right.
}
```

  - C++ syntax: <stream>**.close( )**;

Text file

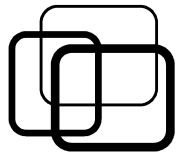


- Read stream:
  - C syntax:
    - **fscanf(**<stream>, “<type format>”, &<var 1>, …**)**;
    - **fgets(**<string>, <max line>, <stream>**)**;

```
FILE  *f = fopen(“C:\\test.txt”, “r”);
if ( !f )  fprintf(stderr, “Error.”);
else {
    int   a, b;        char  s [ 100 ];
    fscanf( f, “%d %d”, &a, &b );
    fgets( s, 100, f );
    fclose( f );
}
```

  - C++ syntax:
    - <stream> **>>** <var>.
    - <stream>.**getline(** <string>, <max line> **)**.

Text file

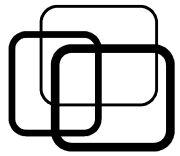


- Write stream:
  - C syntax:
    - **fprintf(**<File stream>, “<Định dạng kiểu>”, <Biến 1>, …**)**;

```
FILE  *f = fopen(“C:\\BaiTap.txt”, “w”);
if ( !f )  fprintf(stderr, “Error.”);
else {
    int      a = 12;
    float   b = 4.165;
    char   c = ‘A’;
    fprintf( f, “Gia tri = %d %f %c”, a, b, c );
    fclose( f );
}
```

  - C++ syntax: <stream> **<<** <var>.

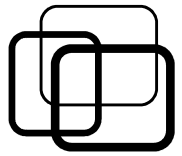
Text file



- File management:
  - Delete file:
    - Syntax: **remove(**“<file path>”**)**;
    - Return: 0 (succeeded), -1 (error).
    - Do not need file stream.

```
if (remove(“C:\\test.txt”) == 0)
        printf(“File deleted.\n”);
else
        printf(“Error: cannot delete file.\n”);
```

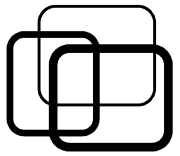
Text file



- File management:
  - Rename/move file:
    - Syntax: **rename(**“<file path>”, “<new file path>”**)**;
    - Return: 0 (succeeded), -1 (error).
    - Both path must be in the same drive.
    - Do not need file stream.

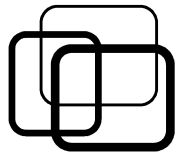
```
if (rename(“C:/test.txt”, “C:/folder\\test2.txt”) == 0)
    printf(“Succeeded.\n”);
else
    printf(“Error.\n”);
```

Summary



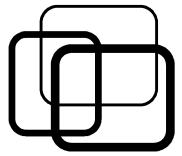
- String:
  - C string:
    - Array of chars + ‘\0’ at the end.
    - Null terminated string.
  - C++ string (std::string):
    - A data type in library <string>.
    - Dynamic string.
  - Input/output:
    - C string: fgets, std::cin.getline, printf, std::cout.
    - std::string: std::getline, std::cout.
  - <ctype.h>: support checking character.

Summary



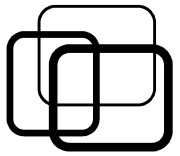
- Text file:
  - File stream: connection between program and file.
  - C file stream: FILE * (<stdio.h>).
    - Open/close: fopen, fclose.
    - Read/write: fscanf, fprintf, fgets.
  - C++ file stream: std::fstream (<fstream>).
    - Open/close: <stream>(), <stream>.close.
    - Read/write: <stream> >>, <<, <stream>.getline.

Summary



- Text file:
  - File stream: connection between program and file.
  - C file stream: FILE * (<stdio.h>).
    - Open/close: fopen, fclose.
    - Read/write: fscanf, fprintf, fgets.
  - C++ file stream: std::fstream (<fstream>).
    - Open/close: <stream>(), <stream>.close.
    - Read/write: <stream> >>, <<, <stream>.getline.

Practice



- ## Practice 6.1:

Write C/C++ program to trim spaces:

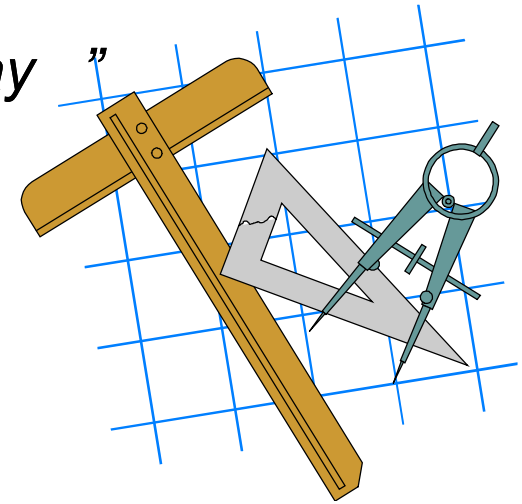
- Enter a sentence of words.
- Delete leading spaces at the beginning (trim left).
- Delete trailing spaces at the end (trim right).
- Delete duplicate spaces between words (keep one space).

Input format:

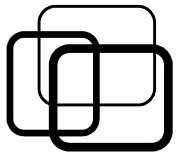
Enter a sentence = “ today is a beautiful day ”

Output format:

“today is a beautiful day”



Practice



- ## Practice 6.2:

  Write C/C++ program as follow:

  - Enter a string S.
  - Count frequencies of each character in S.
  - Print the frequencies in descending order.

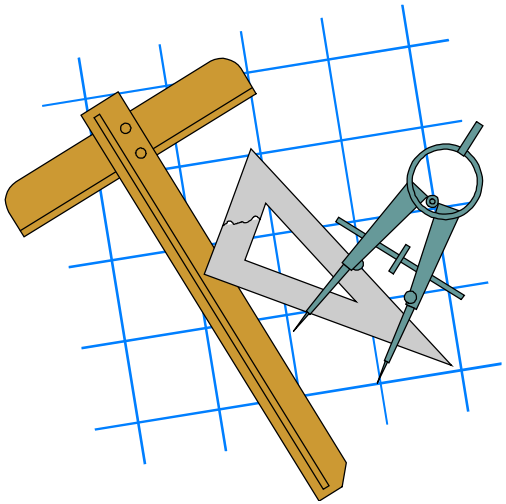
  Input format:

  S = tick tak tok

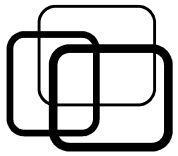
  Output format:

  3: k t

  1: a c i o



Practice



- **Practice 6.3:**

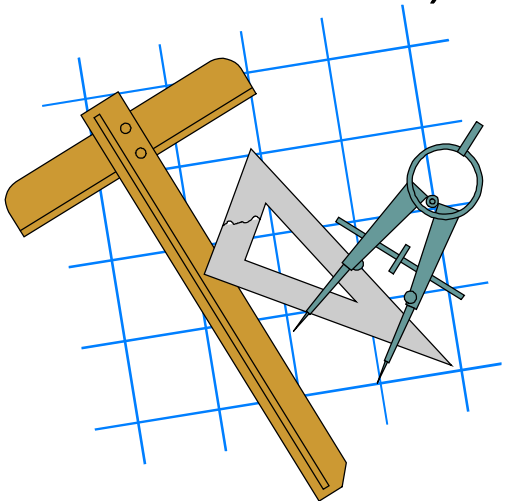
  File MATRIX.TXT stores a matrix of M x N integers:

  - First line: M N (integers, rows and columns).
  - Next M lines, line i stores N elements at row i of matrix.

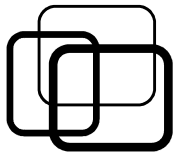
  Write C/C++ program to:

  - Read matrix from MATRIX.TXT.
  - Rotate left the matrix.
  - Write the result to OUTPUT.TXT (same format as MATRIX.TXT).

MATRIX.TXT	OUTPUT.TXT
2 3	3 2
1 2 3	3 6
4 5 6	2 5
	1 4



Practice



- Practice 6.4:

  File NUMBER.TXT stores sequence of integers separated by space.

  Write C/C++ program as follow:

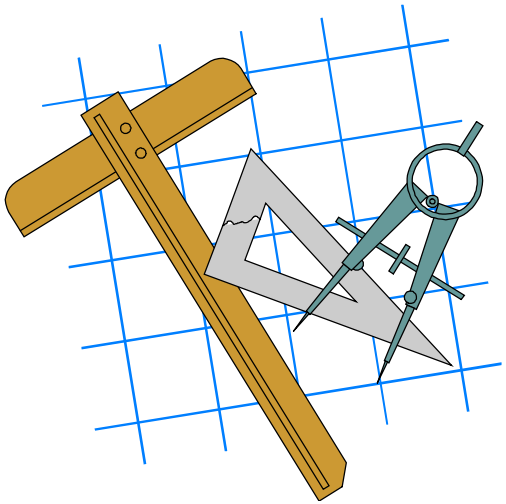
  - Read sequence of integers from NUMBER.TXT.
  - Extract prime numbers from the sequence.
  - Write result to PRIME.TXT (same format as NUMBER.TXT).

  NUMBER.TXT

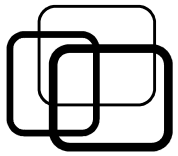
  5 1 2 9 46 23 11 39 43 117 20

  PRIME.TXT

  5 2 23 43 117



Practice



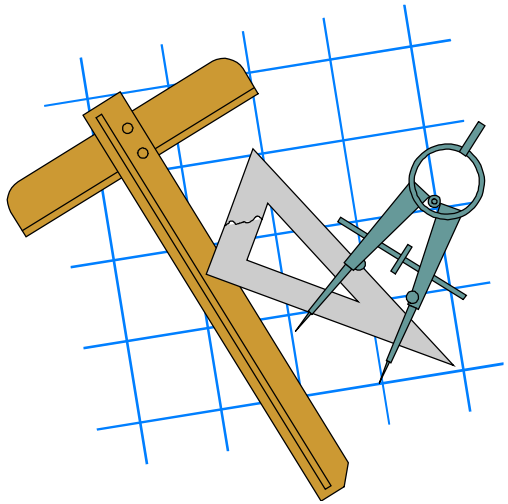
- ## Practice 6.5:

  File OPERATION.TXT stores a list of arithmetic operations, each operation is on a line containing an operator (character +, -, *, /) in between two operands (integers).

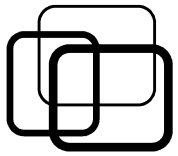
  Write C/C++ program as follow:

  - Read list of operations from OPERATION.TXT.
  - Compute the result of each operation.
  - Write operation results to RESULT.TXT.

OPERATION.TXT	RESULT.TXT
12 + 5	17
2 * 31	62
3 / 0	Divided by zero
60 / 2	30
7 – 20	-13



Practice



- Practice 6.6:

  File STUDENT.TXT stores a list of students, each student is on 2 lines:

  - First line format: student name|student id.
  - Second line format: student points (floats) separated by spaces.

  Write C/C++ program as follow:

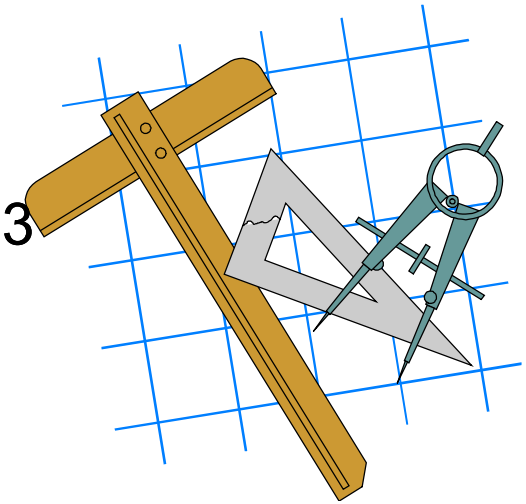
  - Read student list from STUDENT.TXT.
  - For each student, capitalize student name and compute GPA.
  - Write the result to GPA.TXT in descending order of GPA.

STUDENT.TXT

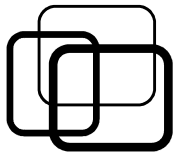
```
nguyen van a|24127001
8.5  6.0  7.5
tran thi b|24127002
9.0  8.0  6.5  7.5
```

GPA.TXT

```
24127002|Tran Thi B|7.6
24127001|Nguyen Van A|7.3
```



Practice



- Practice 6.7:

Write C/C++ program to count words and numbers:

- Read from a text file INPUT.TXT.
- Count and print the number of words and numbers in the file.

Notes:

- A word is a sequence of alphabets.
- A number is a sequence of digits.

INPUT.TXT

-----The, quick ##123 fox,,, 456!!!

Screen output:

Word count: 3.

Number count: 2.