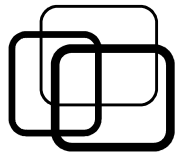
Binary File

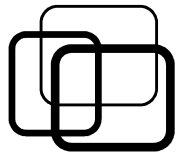
Inst. Nguyễn Minh Huy

Contents



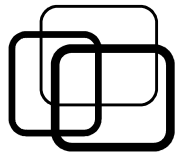
- main() arguments.
- Binary file.

Contents



- **main() arguments.**
- Binary file.

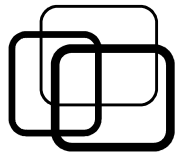
main( ) arguments



- Command-line arguments:
  - Program is a giant function!!
  - How to pass arguments to program?
  - Command-line arguments:
    - Pass arguments to program when calling.
    - main( ) can get the calling arguments.
  - Usage:
    - Run program in command-line terminal.
    - Syntax: <program> **<arg 1> <arg 2> …**

      C:\>BaiTap\baitap1.exe **hello 5 /abc**
      C:\>copy **C:\BaiTap\baitap1.exe D:\Files\baitap1.exe**

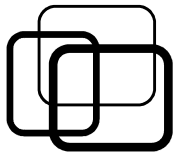
main( ) arguments



- main( ) arguments:
  - Declaration: int main(int **argc**, char ****argv**);
    - argc: argument count .
    - argv: argument variables.
    - Arguments passed as strings.
    - First argument is program name.

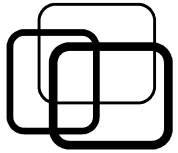
```
int main(int  argc, char  **argv)
{
        cout << “Number of args = “ << argc;
        cout << “Args list:” << endl;
        for (int i = 0; i < argc; i++)
              cout << argv[ i ] << endl;
}
```

Contents

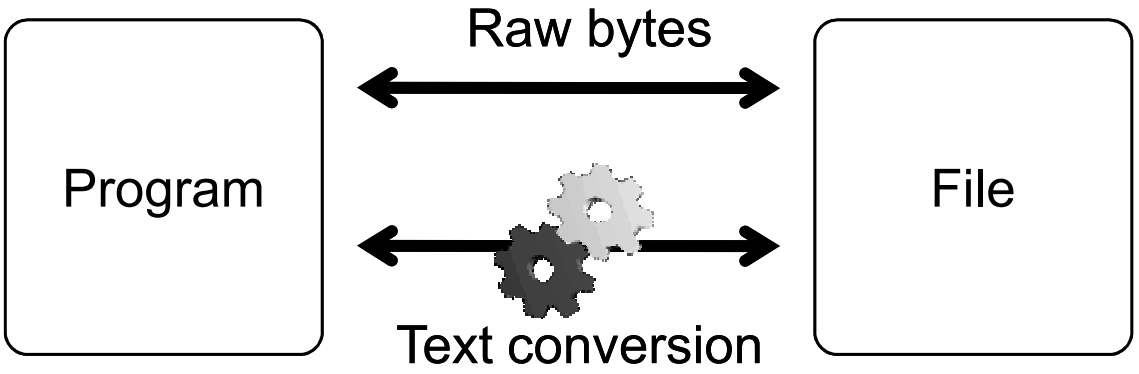


- main() arguments.
- **Binary file.**

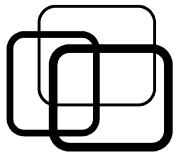
Binary file



- Binary vs. text mode:
  - All file are stored in binary (bit/byte).
  - Binary: read/write raw bytes directly.
  - Text: read/write through conversion (ASCII or Unicode).
  - C/C++ binary mode:
    - C: [r / w / a] **b**.
    - C++: [std::ios::in / out / app] | **std::ios::bin**.



Binary file

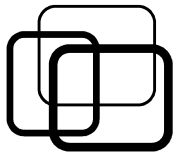


- fread:
  - Read blocks of bytes from file into memory.
    - Syntax: **fread(**<memory pointer>, <block size> , <number of blocks>, <file pointer>**)**;
    - Return: number of read blocks.

    ➔ End of file: number of read blocks < number of blocks.

```
int     x;
char   *p = new char[ 100 ];
FILE  *f = fopen("C:\\BaiTap.txt", "rb");

if ( f != NULL )
{
        fread( &x, sizeof(int), 1, f );     // Read 4 bytes into x.
        fread( p, sizeof(char), 100, f );// Read 100 bytes into p.
        fclose( f );
}
```

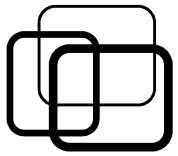
Binary file



- fwrite:
  - Write blocks of bytes from memory into file.
    - Syntax: **fwrite(**<memory pointer>, <block size> , <number of blocks>, <file pointer>**)**;
    - Return: number of written blocks.

```
int      x = 123456;
char   s[ ] = “Hello World”;
FILE  *f = fopen(“C:\\BaiTap.txt”, “wb”);

if ( f != NULL )
{
        fwrite( &x, sizeof(int), 1, f );            // Write 4 bytes x to file.
        fwrite( s, sizeof(char), strlen(s), f );  // Write 11 bytes s to file.
        fclose( f );
}
```

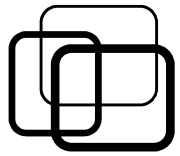
Binary file



- fseek:
  - Move file pointer.
    - Syntax: **fseek(**<file pointer>, <offset>, <origin>**)**;
    - <origin>:
      - SEEK_SET (beginning of file).
      - SEEK_CUR (current position).
      - SEEK_END (end of file).
    - Only works with opening file.

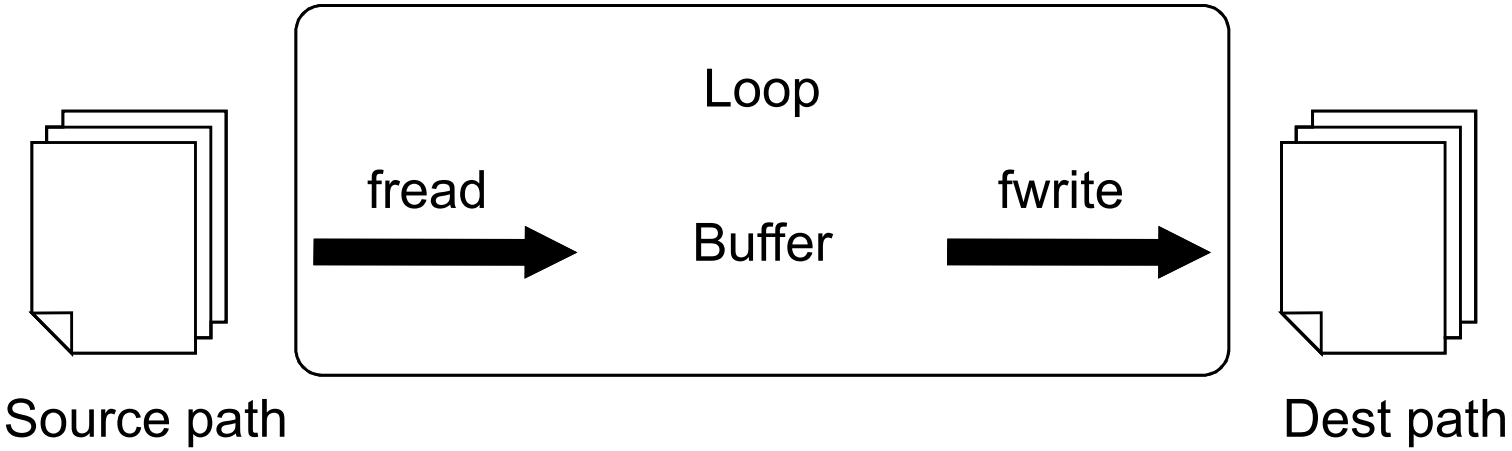
```
FILE  *f = fopen(“C:\\BaiTap.txt”, “r”);
if ( f != NULL )
{
        fseek( f, 2, SEEK_CUR );  // Move forward 2 bytes.
        fclose( f );
}
```

Binary file

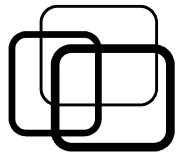


- Practice:
  - Write function to copy file:

    void **copy_file**(<source path>, <dest path).



Binary file

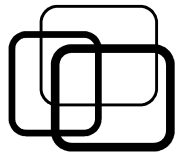


- Read/write struct:
  - Read/write one-by-one bytes of whole struct to file.

    ➔ More effective than read/write struct members.

```
struct Fraction {
        int  num;
        int  denom;
};

void readFraction(FILE *f, Fraction &p) {   // Read 8 bytes from file into p.
        fread( &p, sizeof(Fraction), 1, f );    // First 4 bytes into p.num.
}                                               // Second 4 bytes into p.denom.

void writeFraction(FILE *f, Fraction p) {   // Write 8 bytes p to file.
        fwrite( &p, sizeof(Fraction), 1, f );   // p.num write to first 4 bytes.
}                                               // p.denom write to second 4 bytes.
```

Binary file



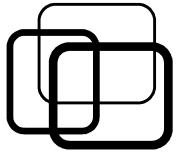
- Library <stdint.h>:
  - What size of int in C?

    ➔ Depends on platform.
  - Binary file read/write needs fix-sized integer.

    ➔ Use <stdint.h>
  - Fix-sized integer:
    - 1 byte: int8_t, uint8_t.
    - 2 bytes: int16_t, uint16_t.
    - 4 bytes: int32_t, uint32_t.
    - 8 bytes: int64_t, uint64_t.

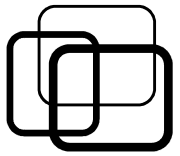
```
#include <stdint.h>

struct Fraction
{
        int32_t  num;
        int32_t  denom;
};
```

Summary



- main() arguments:
  - Calling arguments from command-line terminal.
  - Syntax: int main(int argc, char **argv).
- Binary file:
  - Binary mode: read/write directly.
  - Text mode: read/write through text conversion.
  - C: fread, fwrite, fseek.
  - C++: <stream>.read, <stream>.write, <stream>.seekg.
  - <stdint.h>: fix-sized integers.

Practice



- ## Practice 9.1:

Write C/C++ program named “COPY” to copy file in command line.

Command-line syntax:

- Syntax 1: copy source file to destination file:

  COPY <**source file**> <**destination file**>

- Syntax 2: copy source file to destination path (keep filename):

  COPY <**source file**> <**destination path**>**/**

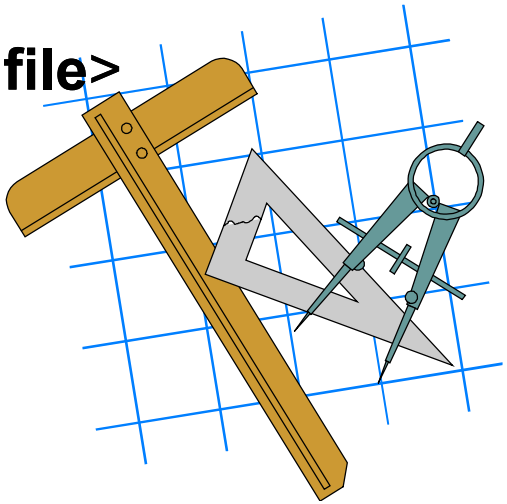
- Syntax 3: join file 1 and file 2 to destination file:

  COPY <**file 1**> + <**file 2**> <**destination file**>

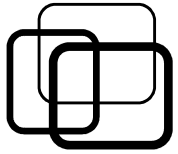
- Syntax 4: show help:

  COPY **-?**

Note: use dynamic string and binary file.



Practice



- Practice 9.2 (*):

  Write C/C++ program to cut Bitmap file into equal parts in command-line. Each part is saved in a new Bitmap file.

  Command-line syntax:

  <program> <Bitmap file> [-h <parts in height>] [-w <parts in width>]

  Example: program cutbmp.exe

  - Cut 3 parts in height (save in 3 new Bitmap files):

    cutbmp.exe d:/images/img1.bmp -h 3

  - Cut 2 parts in height, 4 parts in width (save in 8 new Bitmap files)

    cutbmp.exe d:/images/img1.bmp -h 2 -w 4