Week 6 Lecture 2

NWEN 241
Systems Programming

Alvin C. Valera

alvin.valera@ecs.vuw.ac.nz

Admin Stuff (1)

- Term Test 1 on 7 April @6:10pm
  - Covers Week 1 to Week 5 topics
  - 45 minutes, 45 marks
    - 20 multiple choice questions [20 marks]
    - 10 short answer questions [25 marks]
  - Closed book, permitted materials
    - Only silent non-programmable calculators or silent programmable calculators with their memories cleared are permitted in the test.
    - No electronic dictionaries are allowed.
    - Paper foreign to English language dictionaries are allowed.

Admin Stuff (2)

- No tutorial-style lecture on 7 April

- Assignment #2 (10% of course marks) is due on Wednesday, 20 April 2022 23:59
  - No helpdesk sessions over the break
  - E-mail nwen241-staff@ecs.vuw.ac.nz for help or post question in forum

Content

Continuation of File Stream I/O

- Random Access Operations

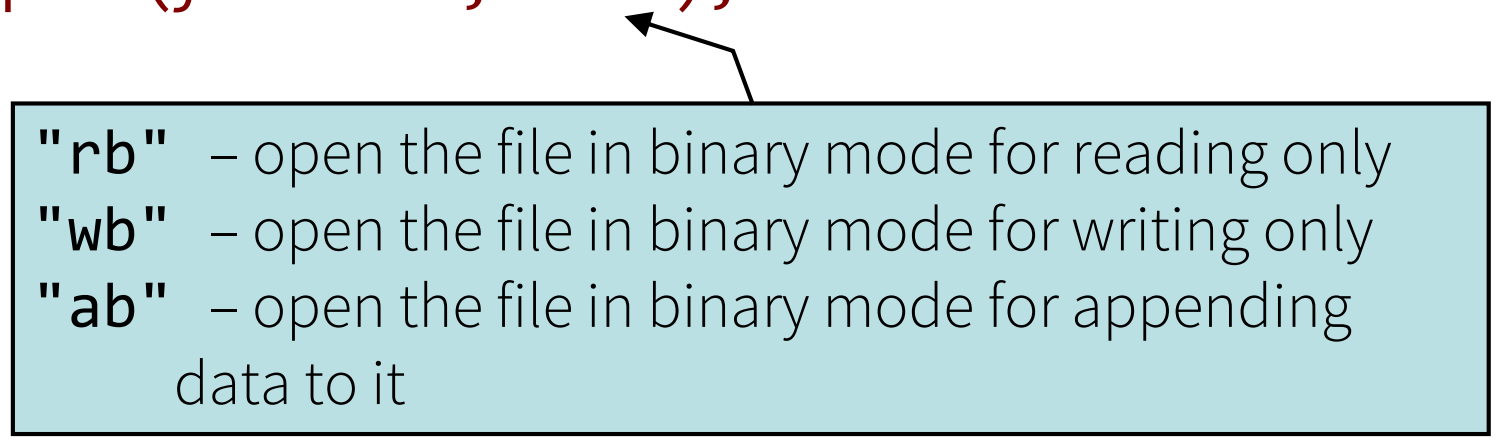
Command Line Arguments

Quick Tutorial on File Stream I/O and Command Line Arguments

Handling Binary Files

- Same as dealing with text files **except in the opening step**

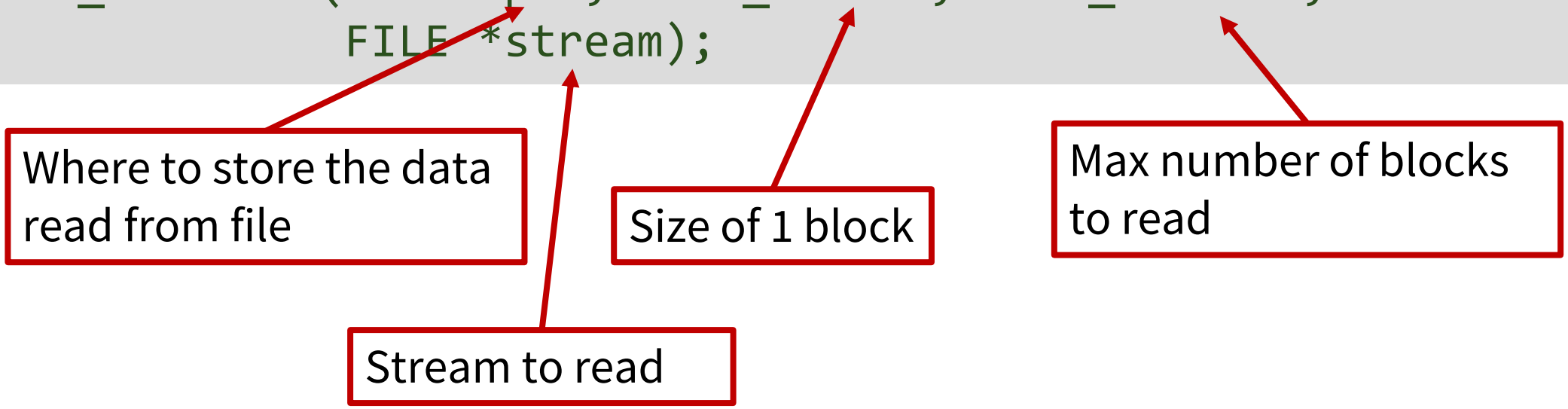
```
FILE *fp; // pointer to stream
        :::
fp = fopen (filename, mode);
```



Reading Binary Files

- Read blocks of binary data from stream

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



- `fread()` returns the actual number of elements read

Example

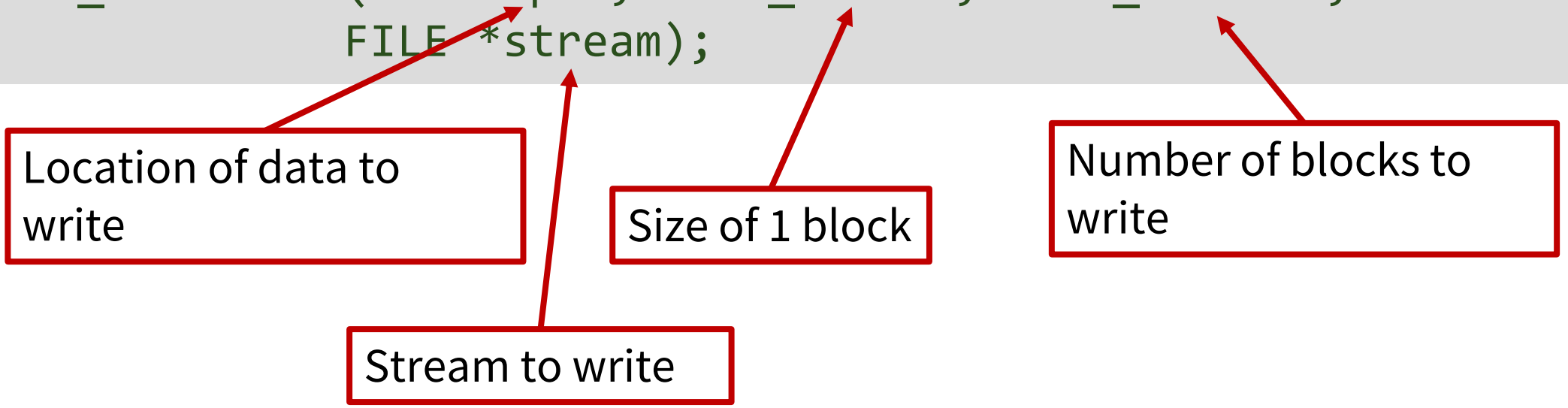
```
FILE *fp;
unsigned char buffer[10];

fp = fopen("file1.exe", "rb");
fread (buffer, sizeof(buffer), 1, fp);
```

- Will read the first 10 bytes of `file1.exe` and store them in `buffer`

Writing Binary Files

- Writes blocks of binary data to stream

```
size_t fwrite (void *ptr, size_t size, size_t nmemb,
               FILE *stream);
```



- `fwrite()` returns the actual number of elements written

Example

```
FILE *fp;
int data[4] = {15, 31, 63, 127};

fp = fopen("datafile", "wb");
fwrite (data, sizeof(int), 4, fp);
```

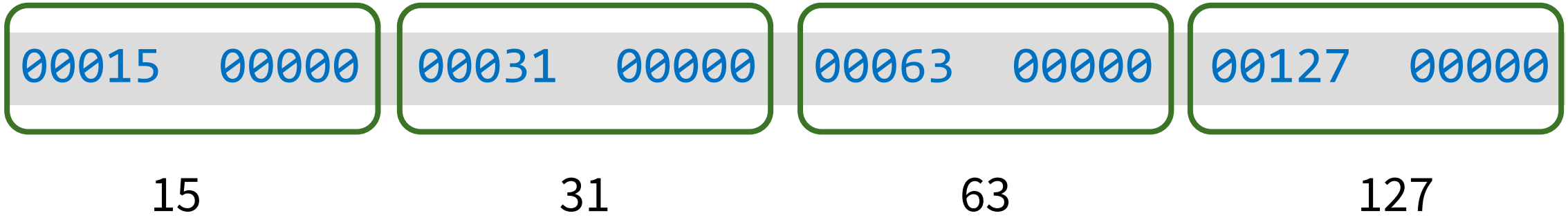
- Will write the `data` array to `datafile`

Example

- In Linux, you can use `hexdump` to view contents of binary file

- `hexdump -d datafile` will display the contents of datafile in decimal

`datafile:`



Random Access

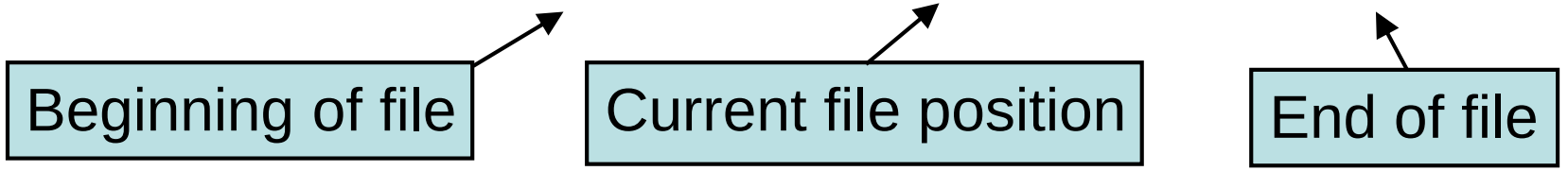
- After opening a file, **read/write position** is either at start or end of file
- To change position, use either `fseek()` or `rewind()`
- To know current position, use `ftell()`

fseek()

- `fseek()` allows repositioning within a file

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

- New position in the file is determined by:
  - `offset` – byte count (possibly -ve) relative to the position specified by `startpoint` where
    - `startpoint: {SEEK_SET, SEEK_CUR, SEEK_END}`



ftell()

- `ftell()` returns the current file position:

```
long ftell(FILE *stream);
```

- This may be saved and later passed to `fseek()`:

```
long file_pos;
file_pos = ftell(fp);
…
fseek(fp, file_pos, SEEK_SET);
/* return to previous position */
```

`rewind()`

- Reposition reading/writing to start of file
- `rewind(fp)` is equivalent to:

```
fseek(fp, 0, SEEK_SET)
```

Command Line Arguments

Command Line Arguments

- *Command line arguments* are parameters supplied to a program when it is invoked

- Example:
  - When invoking the command **`cd`** to change directory, you may have to specify the directory that you want to go to as an **argument**:

```
$ cd /home/yoda/padawan_grades
```

Command line argument

Command Line Arguments

- *Command line arguments* are parameters supplied to a program when it is invoked

- How do these parameters get into the program?

`main()` Function

- The main function can actually be implemented in two ways

```
int main(void)
{
    ...
}
```

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

Command Line Arguments

- *Command line arguments* are parameters supplied to a program when it is invoked

- How do these parameters get into the program?
  - Every C program has a `main()` function
  - `main()` can actually take 2 arguments, conventionally called `argc` and `argv`
  - Command line arguments are passed to the program through `argc` and `argv`

Passing Arguments to `main()`

- General format of command line arguments:

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

  - `argc`
    - Number of arguments (including program name)
  - `argv`
    - Array of strings
    - `argv[0]` → program name
    - `argv[1]` → first argument
    - …
    - `argv[argc-1]` → last argument

Example

- Consider the C program `main_arg.c`:

```c
#include <stdio.h>

int main(int argc, char* argv[])
{
  int i;
  printf("%d arguments\n", argc);
  for(i = 0; i < argc; i++)
    printf("    %d: %s\n", i, argv[i]);
  return 0;
}
```

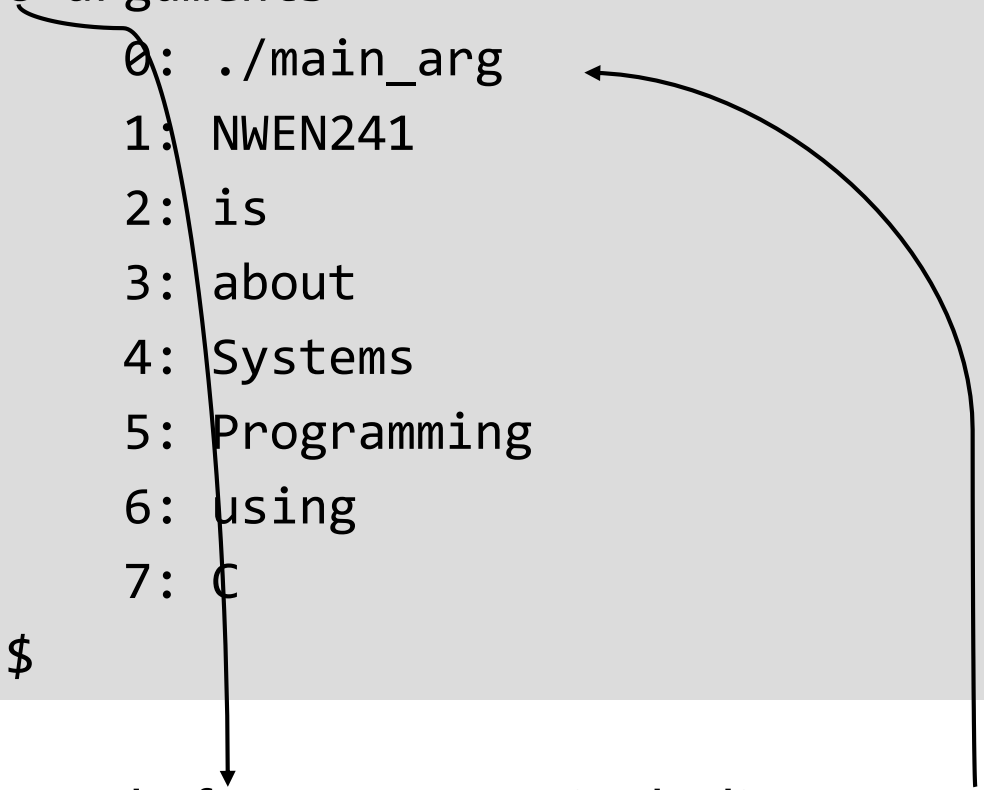
Example

- Compile and generate executable file `main_arg`

```
gcc main_arg.c –o main_arg
```

Program Output

```
$ ./main_arg NWEN241 is about Systems Programming using C
8 arguments
    0: ./main_arg
    1: NWEN241
    2: is
    3: about
    4: Systems
    5: Programming
    6: using
    7: C
$
```



Total of 8 arguments including program name itself.

Arguments are read in as strings.

Next Lecture

- Jyoti Sahni will deliver the second half after the mid-trimester break