Algorithms and Data Structures

Lab 01

Antonio Servetti
Dipartimento di Automatica e Informatica
Politecnico di Torino

Outline

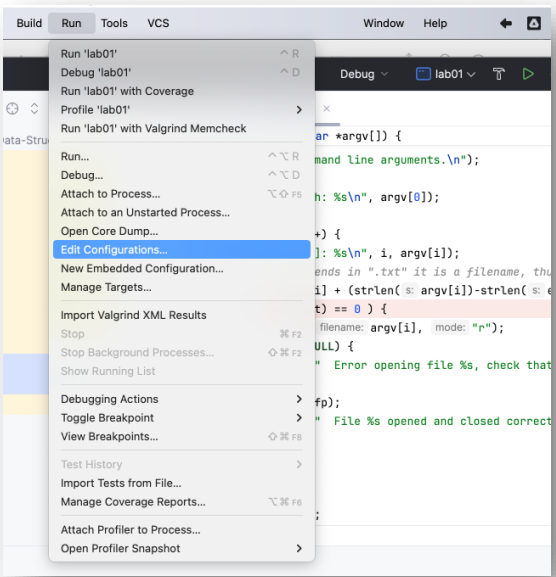
- CLion
  - Create New Project
  - Specify Program Arguments
  - Check Current Working Directory
- Dynamic Allocation
  - Function to read, allocate, and store
  - Elements read from a file
  - Into a dynamically allocated array of structures

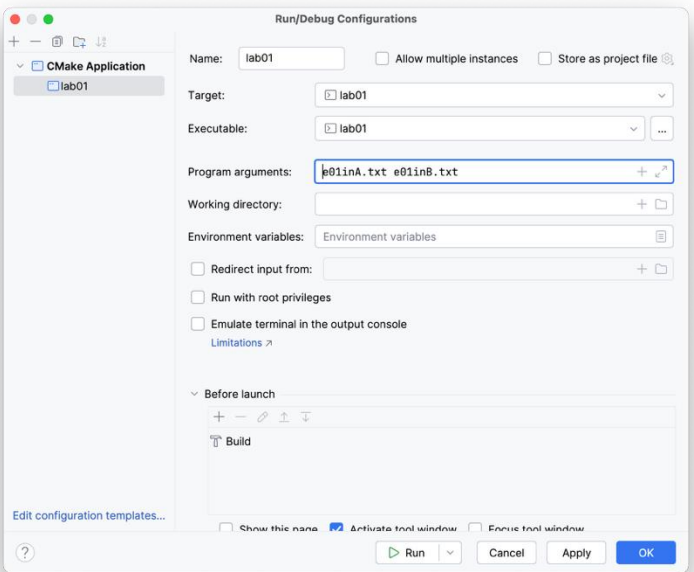
CLion – New Project

- File > New > Project
  - C Executable
    - Set "Location" to a new folder
- Project files and folders
  - main.c – the source file of your program
  - CMakeLists.txt
  - cmake-build-debug/
    - This is the folder where the executable program is built and from where it is executed

CLion – Program arguments

- Run > Edit configurations...
  - Run/Debug Configurations
    - Set "Program arguments" as space separate strings





List program arguments

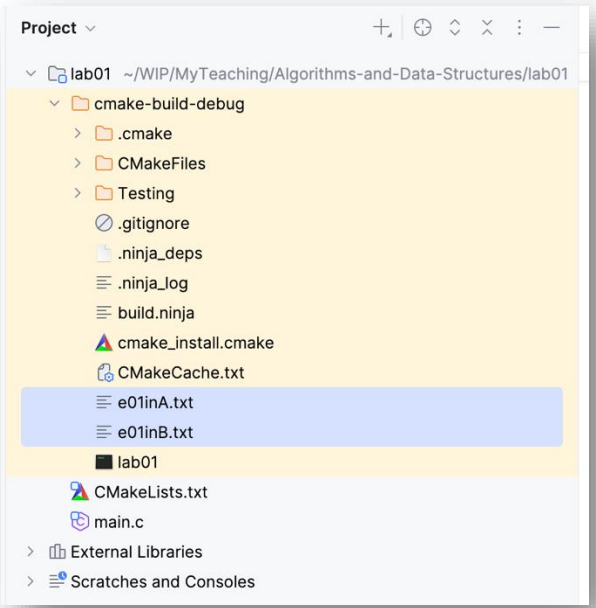
```
printf("Program path: %s\n", argv[0]);
for(i=1; i<argc; i++) {
  printf("argv[%d]: %s\n", i, argv[i]);
  // If argument ends in ".txt" it is a filename
  size_t offset = strlen(argv[i])-strlen(ext);
  char *s = argv[i] + offset;
  if( offset >= 0 && strcmp(s, ext) == 0 ) {
    fp = fopen(argv[i], "r");
    if (fp == NULL) {
      printf("  Error opening file %s.\n", argv[i]);
    } else {
      fclose(fp); printf("  File can be read.\n");
    }
  }
}
```

char ext[] = ".txt"

Try to open file
if argument ends in ".txt"

CLion – Current Working Directory

- CLion executable program is created in subfolder **cmake-build-debug** with the same name of the project and runs from there
- Input files need to be placed into that folder
- That folder is called **cwd** Current Working Directory



Check current working directory

```
// Preprocessor directives
#ifdef _WIN32
  #include <direct.h>
  #define getcwd _getcwd
#else
  #include <unistd.h>
#endif

// Program code (in main body)
char *cwd = getcwd(NULL, 0);
if (cwd != NULL) {
  printf("Current working directory: %s\n", cwd);
  free(cwd);
} else {
  printf("getcwd() error\n");
  return EXIT_FAILURE;
}
```

Dynamic Allocation

- Dynamically allocate an array of structures to store the list of words read from file
- Write a function to read a file, store it in a dynamic array, and return the dynamic array to the caller
  - Specify the no. of elements on the file first row
  - Read this number
  - Allocate the array of the proper size
  - Read the values and store them in the array
  - Use the return statement to return the array pointer from the function

See u02s03-1DArrays.pdf, slide 13 and following ones

Function to read, allocate, store an array

```
int n; struct foo *v;
v = read_and_store(&n);
...
```

Main or client:
Caller

Fortunately, v is
NOT NULL here

Receiving v is useless

```
struct foo *read_and_store (int *n) {
  struct foo *ptr;
  <open and read *n>
  ptr = (struct foo *)
        malloc (*n * sizeof (struct foo));
  if (ptr == NULL)  { ... }
  <read file>
  <close file>
  return ptr;
}
```

Function:
Read & Allocate

Lab01 - Material

- See course material
  - Laboratories > specifications > lab01
- Lab01 assignment
  - lab01.txt
- Sample input files (copy them into CWD)
  - inputfiles/
- These slides
  - lab01.pdf