CSCI 2132
Software Development

Lecture 32:

Dynamically Allocated Arrays

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- Merge Sort with Linked List example finished
- Git and GitLab, comparison to Subversion (svn)
- File Manipulation in C
  - streams and files

File Pointers

- In C, streams are accessed through *file pointers*; e.g.:

  ```
  FILE *fp;
  ```

- Defined in `stdio.h`
- When a file is open, a file pointer has address of a file descriptor structure
- Two types of files: text and binary
- Text files:
  - newline character may be treated specially
  - may have a special marker byte at the end
- Binary files: raw access

Opening a File

- Using function `fopen`:

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

- Modes for text files:
  - `"r"` to open for reading,
  - `"w"` to open for writing (deletes old contents),
  - `"a"` to open for appending,
  - `"r+"` reading and writing, starts at beginning,
  - `"w+"` reading and writing, deletes old contents,
  - `"a+"` reading and writing, writes at the end position.

Opening a Binary File

- `"rb"` to open for reading,
- `"wb"` to open for writing,
- `"ab"` to open for appending,
- `"r+b"` or `"rb+"` reading and writing, starts at beginning,
- `"w+b"` or `"wb+"` reading and writing, deletes old contents,
- `"a+b"` or `"ab+"` reading and writing, writes at the end position.
- Binary and text files are not treated differently on a Unix system
- `fopen` returns NULL if unsuccessful

Closing a File

- To close a file:

```
int fclose(FILE* fp);
```

- Returns 0 if successful, otherwise returns EOF

Formatted I/O with a File

- Two main functions:

```
int fprintf(FILE *stream, const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);
```

- `printf(...)` is equivalent to `fprintf(stdout, ...)`
- `scanf(...)` is equivalent to `fscanf(stdin, ...)`
- Printing an error message: `fprintf(stderr, ...)`

Example with stderr

```
FILE *fp;
fp = fopen("hello.txt", "w");
if (fp == NULL) {
   fprintf(stderr, "Cannot open hello.txt");
   exit(EXIT_FAILURE);
}

fprintf(fp, "hello, world\n");
fclose(fp);
```

Character I/O

- Some interesting functions:

```
int putc(int c, FILE *stream);
int getc(FILE *stream);
```

- Similar to `putchar` and `getchar`
- Fill-in-the-blanks example:

```
~prof2132/public/countchar.c-blanks
```

More Read and Write Functions

- Reading end-of-file indicator:

```
int feof(FILE *stream);
```

- Reading and writing blocks of data:

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

- Note: we can ignore keyword `restrict`

File Positioning

- Reading and setting file position:

```
void rewind(FILE *stream);
long int ftell(FILE *stream);
int fseek(FILE *stream, long int offset,
          int whence);
```

- `whence` can be `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`
- Additional functions (better for very large files):

```
int fgetpos(FILE *restrict stream,
            fpos_t *restrict pos);
int fsetpos(FILE *stream,
            const fpos_t *pos);
```

Example: Writing and Reading Double in Binary

```
/* Program: writedouble.c */
#include <stdio.h>

int main() {
  FILE *fp;
  double d;
  printf("Enter a double: "); scanf("%lf", &d);
  printf("Saving double in tmp1.\n");
  fp = fopen("tmp1", "wb");
  fwrite(&d, sizeof(double), 1, fp);
  close(fp);
  return 1;
}
```

```
/* Program: readdouble.c */
#include <stdio.h>

int main() {
  FILE *fp;
  double d;
  int c;
  printf("Reading double from tmp1:\n");
  fp = fopen("tmp1", "rb");
  fread(&d, sizeof(double), 1, fp);
  close(fp);
  printf("Read: %lf\n", d);
```

```
  printf("Bit layout:\n");
  rewind(fp);
  while(EOF != (c = getc(fp))) {
    int b = 1 << 7;
    for (; b != 0; b >>= 1)
      putchar( (b & c) ? '1' : '0' );
  }
  putchar('\n');

  return 1;
}
```