H1 vg101: Introduction to Computer Programming

H2 RC 6

CHEN Xiwen

2019/6/28 (FRI)

H3 C-Strings

- Declaration and Initialization `<demoCString.c>`
  1. Array-style:

     ```
     1  char s1[10] = "hello";
     2  char s2[] = "hello";
     ```

     Q: What if we declare `s2` without initialization? i.e., `char s2[];`?

  2. Pointer-style:

     ```
     1  char* s3 = malloc(10 * sizeof(char));
     2  memset(s3, 'a', 9);
     3  free(s3);
     4  // char* s4 = "hello";
     ```

- Assignment `<demoAssign.c>`

  ```
  1  char s1[10];
  2  char s2[];
  3  s1 = "hello"; // error: array type cannot be assigned;
  4  s2 = "hello"; // error: fail on declaration in the first place;
  5                //        size missing;
  6
  7  char* s3 = malloc(10 * sizeof(char));
  8  s3 = "hello"; // not okay;
  9  *s3 = "hello"; // error: *s3 represents s3[0] now, a single char;
  ```

- Read string from `stdin` `<demoStdin.c>`
  1. `int scanf(const char* format, ...)`: formatted input (`%s`: read until a space is met, because it only matches non-white-space characters)
  2. `int getc(FILE* stream) / int getchar()`: read single character, use `getc(stdin)` to read from `stdin`
  3. `char* gets(char* str)`: get a line (read until `\n` is met, `\n` *will not* be included) [deprecated from `C11`]
  4. `fgets(char* str, int size, FILE* stream)`: read from stream for at most `size` chars (read until end-of-file or `\n` is met, `\n` *will* be included, use `fgets(str, size, stdin)` to read from `stdin`)
- Print string to `stdin` `<demoStdin.c>`

1. `int putc(int ch, FILE* stream);` use `putc(ch, stdin)` to write `ch` to `stdin`
2. `int puts(char* str)`: append a `\n` at the end of the string
3. `int printf(const char* format, ...)`: formatted output

- Conversion

1. `int atoi(const char* str);`
2. `double atof(const char* str);`
3. `long atol(const char* str);`

- Other operations

1. `size_t strlen(const char* s);`
2. `char* strcpy(char* dst, const char* src);`
3. `char* strcat(char* restrict s1, const char* restrict s2);`
4. `int strcmp(const char* s1, const char* s2);`
5. `char* strchr(const char* s, int c);` locate the first occurrence of `c` as a `char`
6. `char* strstr(const char* haystack, const char* needle);` locate the first occurrence of the null-terminated string `needle` in the null-terminated string `haystack`.

RULE OF THUMB: look for the function specifics in references.

H3 File I/O

- Recall in MATLAB, we use `fid`, and in `C`, we use a "file pointer" `FILE*`
- Open a file

  `FILE* fp = fopen(filename, mode);`

Mode	Explanation
`r`	read from **existing** file
`w`	open or **create** a file to write
`a`	open or **create** a file, append data
`r+`	read from and write to an **existing** file
`w+`	open or **create** a file to read and write, discarding existing contents
`a+`	open or **create** a file to read and write, append to existing contents

- Close a file: `fclose(fp);`
- Read from a file `<demoFileIO.c>`

1. `int fscanf(FILE* restrict stream, const char* format, ...);`
2. `int fgetc(FILE* stream);`
3. `char* fgets(char* restrict str, int count, FILE* restrict stream);`

   Read at most `count - 1` characters from the given file stream and store them in the character array `str`.

- Write to a file `<demoFileIO.c>`

1. `int fprintf(FILE* restrict stream, const char* restrict format, ...);`
2. `int fputc(int ch, FILE* stream);`
3. `int fputs(const char* restrict str, FILE* restrict stream);`

- Move file position indicator `<demoFileIO.c>`
    1. `void rewind(FILE* stream);` move to the beginning of file
    2. `long ftell(FILE* stream);` return the file position indicator
    3. `int fseek(FILE* stream, long offset, int origin);` set the file position indicator for the file stream to the value pointed by `offset`
        - Beginning: `SEEK_SET`
        - Current: `SEEK_CUR`
        - End: `SEEK_END`

Command Line Arguments

- `int argc`: the number of arguments
- `char* argv[]`: an array of strings representing input, the first argument is the program name, e.g., if you run with

    ```
    $ ./program -h
    ```

    then `argv[0] == "./program"`.
- Arguments need to be parsed, and change the values of the variable in the program.
- Example: `<demoCmdArg.c>`

> // create an array, and print out in a style specified by `verbose` argument;
>
> /*
>
> suppose there are the following available arguments:
>
> -h | --help print this help message
>
> -n n | --length=n length of array
>
> -v | --verbose verbose output
>
> -r | --random randomly set elements, or by increment
>
> */

Circular Doubly Linked List in Practice

- `create_list`
- `insert`
- `delete`

Advantages of circular doubly linked list:

- Easy to insert or remove elements
- Easy to preserve order