Functions, File Handing

Data and File Structures Laboratory

http://www.isical.ac.in/~dfslab/2017/index.html

Basics

- "Sub"-program
  e.g., `gcd`, `area of a triangle`
- Usually takes 1 or more *arguments* (input values)
- May compute a *return value* (output)

Writing functions

Syntax

- `return_type function_name (argument1_type argument1, argument2_type argument2, ... )`

```
{
    /* function body (local variables + statements) */
}
```

- `return` statement
  - `return;` or
  - `return expression;`
- `void`: special keyword to denote no arguments or return value

Using functions

Syntax

- In an expression:
  `x = ... * function_name(input1, input2, ...) - ... ;`
- Only for the effect: `function_name(input1, input2, ...);`
- Ordering, number and types of inputs given to a function call *must match* with the ordering, number and types of arguments in the function header.

Prototypes

Prototype

```
return_type function_name (argument1_type argument1,
            argument2_type argument2,
            ... );
```

- Used to “announce” functions before they are actually written
- Argument types have to be specified, but argument names are optional
- Note the semicolon at the end

Built-in libraries

- **Mathematical functions:** `#include <math.h>`
- **Character types:** `#include <ctype.h>`
- **String functions:** `#include <string.h>`
- **Miscellaneous functions:** `#include <stdlib.h>`

Look up the man pages!

Scope of variables

- Variable names defined within functions are *local* variables
  - not *visible* (cannot be used) outside the function
  - hides / masks any *global* variable with the same name
- *Call by value:* a copy is made of any variable that is passed as an input argument to a function
  - changes to the variable inside the function are not visible outside
  - **exception:** *arrays!*

Arrays and functions

```
int f ( int A[100] , int size )
{
    ...
}
```

Arrays and functions

```
int f ( int A[100] , int size )
{
    ...
}
```

OR

```
int f ( int A[] , int size )
{
    ...
}
```

Structures and functions

- If a large structure is to be passed to a function, it is generally more efficient to pass a pointer than to copy the whole structure.
- Similarly for return values (**but be careful!**)

File handling

Opening a file:

`fopen(filename, mode)`

`filename`: string (`char *`) containing name of file

`mode`: string specifying whether file is to be opened in read/write mode

- `"r"`, `"w"`, `"a"`: read mode, write mode, append mode
- `"r+"`, `"w+"`, `"a+"`: read/write mode, write/read mode, read/append mode (see man page for more details)

Example:

```
FILE *fp;
if (NULL == (fp = fopen("a.txt", "r")))
   ERR_MESG("Error opening file");
```

Closing a file:

`fclose(fp);`

File handling

Reading / writing text:

`fgetc(fp)`: reads and returns the next character from `fp`, or EOF on end of file or error

Typical usage: `while (EOF != (c = fgetc(fp))) ...`

`fgets(s, n, fp)`: reads at most `n-1` characters or one line (whichever is shorter), stores input in character array `s` and terminates `s` using `'\0'`;
returns `s` or `NULL` on end of file (i.e., there is nothing to be read) or error

Typical usage: `while (NULL != fgets(s, n, fp)) ...`

`fputc(c, fp)`: writes `c` to `fp`

`fputs(s, fp)`: writes string `s` to `fp`

File handling

Reading / writing data:

`fread((void *) buffer, sz, n, fp)`: reads `n` elements of data, each of size `sz` bytes from `fp`, stores them in `buffer`; returns number of elements read.

`fwrite((void *) buffer, sz, n, fp)`: writes `n` elements of data from `buffer`, each of size `sz` bytes to `fp`; returns number of elements written.

Acknowledgements

- `http://cse.iitkgp.ac.in/~pds/notes/`
  (please see the above page for many more practice problems)