

Other standard input/output functions in C

Standard Input/Output functions in C

So far, we have extensively used two input/output functions

```
scanf ("%format", &variable_name);  
printf ("%format", variable_name);
```

There are several other functions in C to perform input/output operations.

Standard Input/Output functions in C: getchar() and putchar()

- `getchar()` gets a character from standard input.
- `putchar()` writes a character to standard output.

```
int main () {  
    char c;  
  
    printf("Enter a character: ");  
    c = getchar();  
  
    printf("Character entered: ");  
    putchar(c);  
  
    return(0);  
}
```

Program output
Enter a character: B
Character entered: B

Standard Input/Output functions in C: gets() and puts()

- gets () reads a newline-terminated string from standard input and writes into the specified string.
- puts () writes the string a trailing newline to standard output.

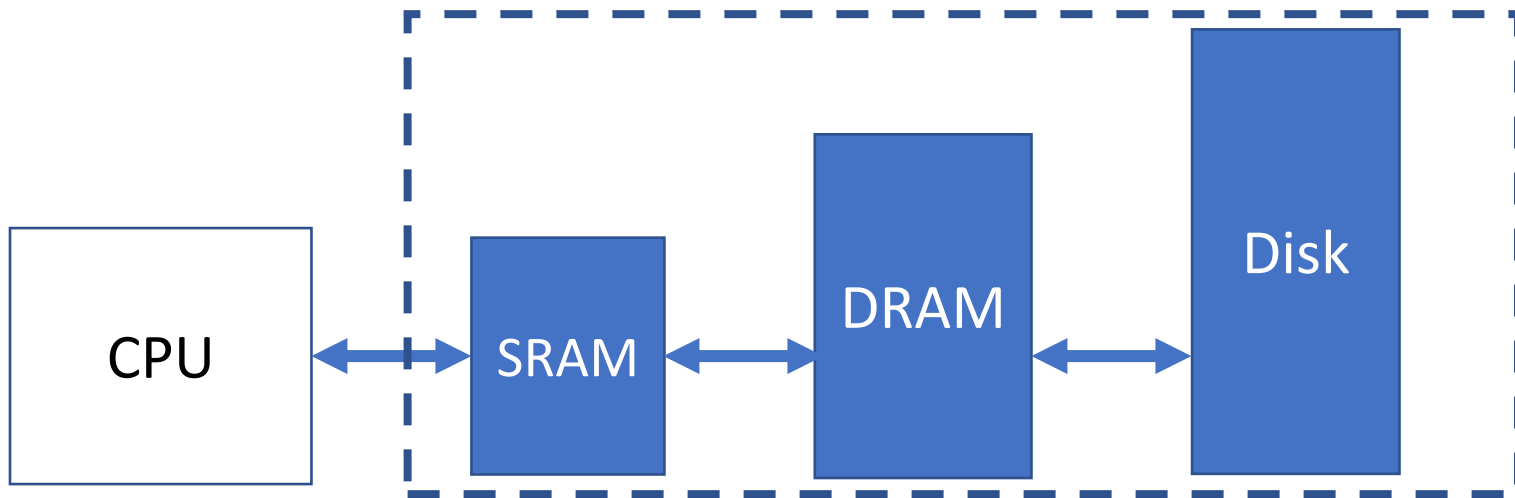
```
int main(){  
    // char array of length 100  
    char str[100];  
    printf("Enter a string: ");  
    gets( str );  
    puts( str );  
    return 0;  
}
```

Program output
Enter a string: ABC DEF
ABC DEF

File handline in C

Use of files in program

- Large data volumes
- E.g. data from statistics, experiments, human genome, population records etc.



File resides in Disk
→ File read/write are slow

Opening file from a C program

- For opening a file, `fopen()` function is used with the required access modes.

```
FILE *fp; /*variable fp is pointer to type FILE*/
```

```
fp = fopen("filename", "mode");
```

```
/*opens file with name filename , assigns identifier to fp */
```

`fopen()` returns NULL, if it is unable to open the specified file.

- File pointer `fp` points to the 'file' resource
 - contains all information about file
 - Communication link between system and program
- An opened file is closed by passing the file pointer to `fclose()`

```
fclose(fp);
```

Different modes

- Reading mode (**r**)
 - if the file already exists then it is opened as *read-only*
 - sets up a pointer which points to the first character in it.
 - else error occurs.
- Writing mode (**w**)
 - if the file already exists then it is *overwritten* by a new file
 - else a new file with specified name created
- Appending mode (**a**)
 - if the file already exists then it is opened
 - else new file created
 - sets up a pointer that points to the last character in it
 - any new content is appended after existing content

Additional modes

- r+ opens file for both reading/writing an *existing* file
 - doesn't delete the content of if the *existing* file
- w+ opens file for reading and writing a *new* file
 - Overwrites if the specified file already exists
- a+ open file for reading and writing from the last character in the file

Functions for reading or writing: `getc()` and `putc()`

There are several functions to read from and write to a file.

- `getc()` – read a `char` from a file.
- `putc()` – write a `char` to a file

Functions for reading or writing: getc() and putc()

```
int main(){
    char ch;
    FILE *fp0, *fp1;
    fp0 = fopen("infile.txt", "r");
    fp1 = fopen("outfile.txt", "w");
    if (fp0==NULL || fp1==NULL){
        printf("Cannot open files\n");
        exit(-1);
    }

    ch = getc(fp0); // reads one char from first file
    while (ch != EOF){ // EOF is 'end of file'
        printf("%c", ch); // Displays on screen
        putc(ch, fp1); // writes to second file
        ch = getc(fp0); // reads another char from first file
    }
    fclose(fp0);
    fclose(fp1);
    return 0;
}
```

Functions for reading or writing: fprintf() and fscanf()

- Similar to printf() and scanf()
- in addition the file pointer is provide as an input
- Examples:

To read one `int` from a file with file pointer `fp0`

```
int i;  
fp0=fopen("some_file", "r");  
fscanf(fp0, "%d", &i);
```

To write one `int` to a file with file pointer `fp1`

```
int i=4;  
fp1=fopen("some_file", "w");  
fprintf(f1, "%d", i);
```

Functions for reading or writing: fprintf() and fscanf()

```
int main(){
    char ch;
    FILE *fp0, *fp1;
    fp0 = fopen("infile.txt", "r");
    fp1 = fopen("outfile.txt", "w");
    if (fp0==NULL || fp1==NULL){
        printf("Cannot open files\n");
        exit(-1);
    }

    fscanf(fp0, "%c", &ch); // reads one char from first file
    while (ch != EOF){      // EOF is 'end of file'
        printf("%c", ch);   // Displays on screen
        fprintf(fp1, "%c", ch); // writes to the second file
        fscanf(fp0, "%c", &ch); // reads another char from first file
    }
    fclose(fp0);
    fclose(fp1);
    return 0;
}
```

Typical file errors

Typically, errors happen when a program

- tries to read beyond end-of-file (EOF)
- tries to use a file that has not been opened
- performs operation on file not permitted by 'fopen' mode

Example:

```
fp=fopen("filename", "r");  
...  
fprintf(fp, "%d", i);
```

- opens file with invalid filename
 - writes to write-protected file
- Example: files with read-only permission

Handling these errors

Programmer can perform the following checks

- `feof(fp)` returns a non-zero value when End-of-File is reached, else it returns zero

```
fp = fopen("somefile", "somemode");  
...  
if(feof(fp)){  
    printf("End of file\n");  
}
```

- `ferror(fp)` returns nonzero value if error detected else returns zero

```
fp = fopen("somefile", "somemode");  
...  
if(ferror(fp) != 0)  
    printf("An error has occurred\n");
```

Random access to file [Optional slide]

Data in a file is basically a collection of bytes.

We can ***directly*** jump to a target byte-number in a file without reading previous data using `fseek()`

- Syntax: `fseek(file-pointer, offset, position);`
- position: 0 (beginning), 1 (current), 2 (end)
- offset: number of locations to move from specified position

Examples:

```
fseek(fp, -m, 1); // move back by m bytes from current  
fseek(fp, m, 0); // move to (m+1)th byte in file
```

- `ftell(fp)` returns current byte position in file
- `rewind(fp)` resets position to start of file

Random access to file [Optional slide]

```
int main () {  
    FILE *fp;  
  
    fp = fopen("file.txt","w+");  
    fprintf(fp,"%s", "This is something");  
  
    fseek( fp, 7, 0);  
    fprintf(fp,"%s"," C Language");  
    fclose(fp);  
  
    return(0);  
}
```

The program

1. writes “This is something”
2. then moves to 7 byte-positions after beginning (i.e., 8th position)
3. writes " C Language" (overwriting any data that exists)

Thus, the final content is “This is C Language”

Command line arguments in C

Command line arguments in C

We can modify a program to receive arguments from command line.

Example

```
./a.out string1 string2 string3
```

Syntax is

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

- argc (Argument Count) is **int** and *automatically* stores the number of command-line arguments passed by the user including name of program.
- argv (Argument Vector) is array of **char** pointers listing all the arguments.
argv[0] is the name of the program
argv[1] is the first command-line argument etc.

Command line arguments in C

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

```
$/a.out how are you?  
You have entered 4 arguments:  
a.out  
how  
are  
you?
```

Another example: Command line arguments in C

```
#include <stdio.h>
#include <stdlib.h>
#define PI 3.1416
int main (int argc, char *argv[]){
    double r, area, circ;

    char *a = argv[1];
    int diameter = atoi(a);

    if(argc>1)
        printf("You have entered %d\n",diameter);
    else{
        printf("Enter diameter:");
        scanf("%d", &diameter);
    }

    r= diameter/2;
    area = PI*r*r;
    circ= 2*PI*r;

    printf ("Circle with diameter %d\n", diameter);
    printf ("has area of %f\n", area);
    printf ("and circumference of %f\n", circ);

    return (0);
}
```

atoi() converts a string to an integer.

```
$/a.out
Enter diameter:2
Circle with diameter 2
has area of 3.141600
and circumference of 6.283200
```

```
$/a.out 2
You have entered 2
Circle with diameter 2
has area of 3.141600
and circumference of 6.283200
```