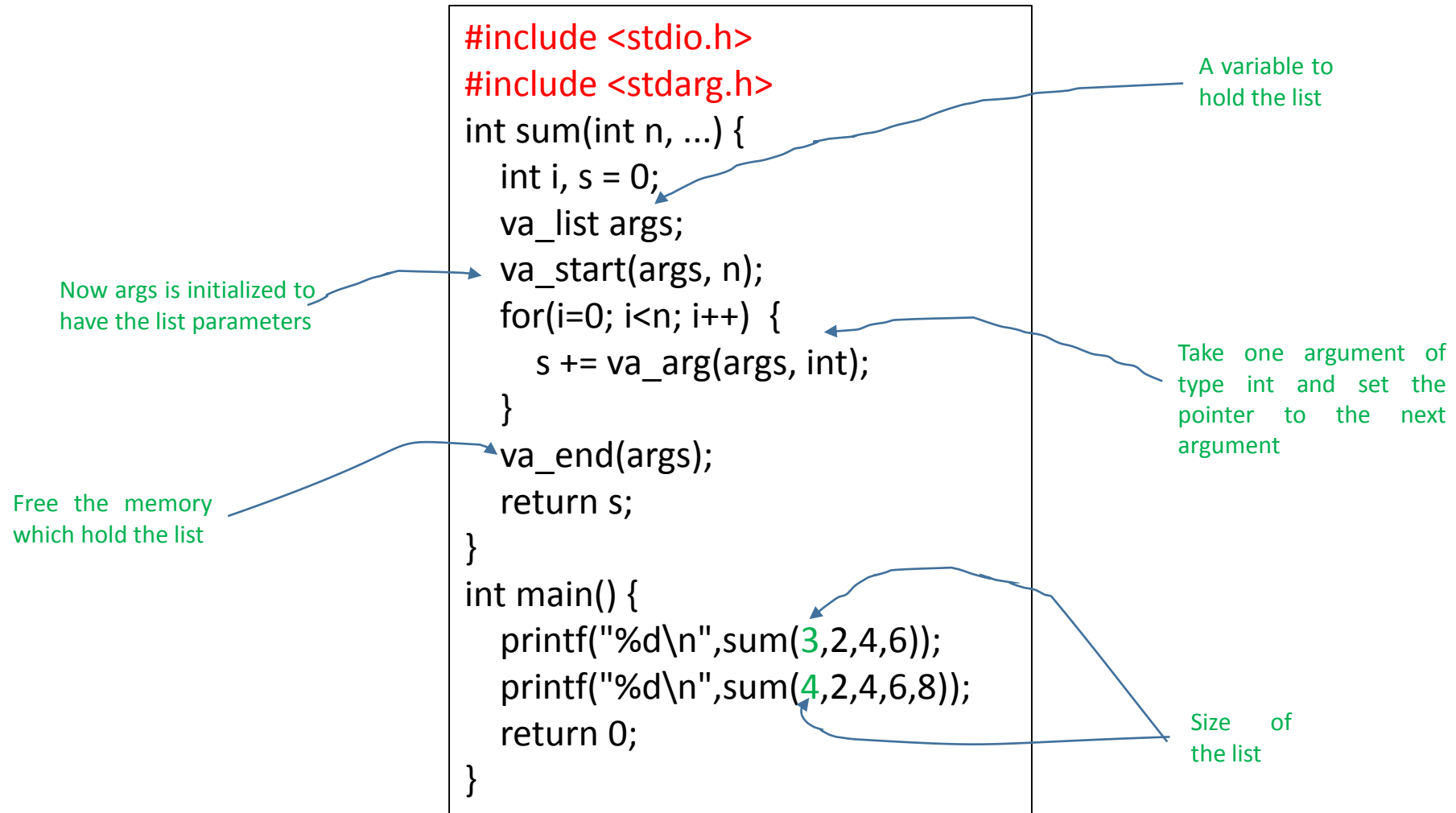


# Variable Arguments

## Ellipsis

- ✓ allow a function to receive undefined number of parameters using defined macros declared in stdarg.h.
- ✓ **va\_list**: a variable to hold the list.
- ✓ **va\_start**: macro to initialize the **va\_list** variable to an argument list.
- ✓ **va\_arg**: macro to access each item in argument list.
- ✓ **va\_end**: macro to clean up the memory assigned to **va\_list** variable.



```
#include <stdio.h>
#include <stdarg.h>
typedef struct Course{
    char name[25];
    double score;
} Course;
double avg(int n, ...){
    double a = 0;
    int i;
    va_list args;
    va_start(args, n);
    for(i=0; i<n; i++){
        Course x = va_arg(args, Course);
        a += x.score;
    }
    va_end(args);
    return a/n;
}
int main(){
    Course c1 = {"Java",100};
    Course c2 = {"C",90};
    Course c3 = {"History",86};
    printf("%s\n",avg(3,c1,c2,c3));
    return 0;
}
```

Take one argument of type struct and set the pointer to the next argument

```
#include <stdio.h>
#include <stdarg.h>
#include <string.h>
void stam(int n, ...)
{
    int age;
    char name[10];
    va_list args;
    va_start(args, n);
    strcpy(name, va_arg(args, char*));
    age = va_arg(args, int);
    printf("My name is %s\n", name);
    printf("Age %d\n", age);
    va_end(args);
}
int main()
{
    stam(2, "Alex", 35);
    return 0;
}
```

- Data types of a function parameters are being validated by the compiler. However, in case of ellipses it is not.
- Ellipses is not safe as the sent values can be of different types than expected.

**Output:**  
My name is Alex  
Age 35

```
int sum(int n, ...)
{
    int i,s=0;
    va_list args;
    va_start(args,n);

    for(i=0;i<n;i++)
    {
        s += va_arg(args,int);
    }
    va_end(args);
    return s;
}
```

```
int mult(int n, ...)
{
    int i,s=1;
    va_list args;
    va_start(args,n);

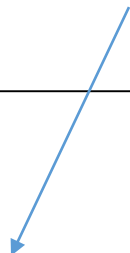
    for(i=0;i<n;i++)
    {
        s *= va_arg(args,int);
    }
    va_end(args);
    return s;
}
```

Pointer to ellipses function



```
int main(void)
{
    int (*pf)(int,...);
    pf=sum;
    printf("%d\n",pf(2,2,3));
    pf=mult;
    printf("%d\n",pf(2,2,3));
    return 0;
}
```

Array of pointers  
to ellipses function



```
int main(void)
{
    int i;
    int (*pf[2])(int,...);
    pf[0]=sum;
    pf[1]=mult;

    for(i=0;i<2;i++)
    {
        printf("%d\n",pf[i](2,2,3));
    }
    return 0;
}
```

```
int getchar(void)
int putchar(int)
```

- ✓ An input/output mechanism which allow to read/write one character at a time from/to the *standard input* (normally the keyboard).
- ✓ getchar returns the character read as an unsigned char cast to an int or EOF on end of file or error.
- ✓ getchar is equivalent to **getc** with stdin as its argument.
- ✓ putchar returns the character written as an unsigned char cast to an int or EOF on error.

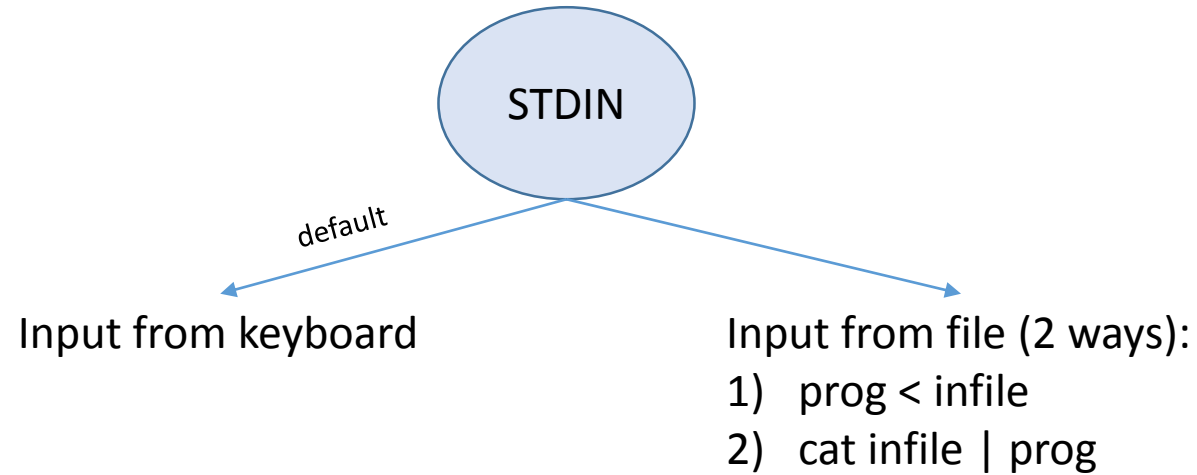
```
#include <stdio.h>
#include <ctype.h>
int main()
{
    int c;
    while ((c = getchar()) != EOF)
        putchar(tolower(c));
    return 0;
}
```

prog.c

CTRL-D/CTRL-DD = EOF in Unix system  
CTRL-Z = EOF in Win system

```
#include <stdio.h>
#include <ctype.h>
int main()
{
    int c;
    while ((c = getchar()) != EOF)
        putchar(tolower(c));
    return 0;
}
```

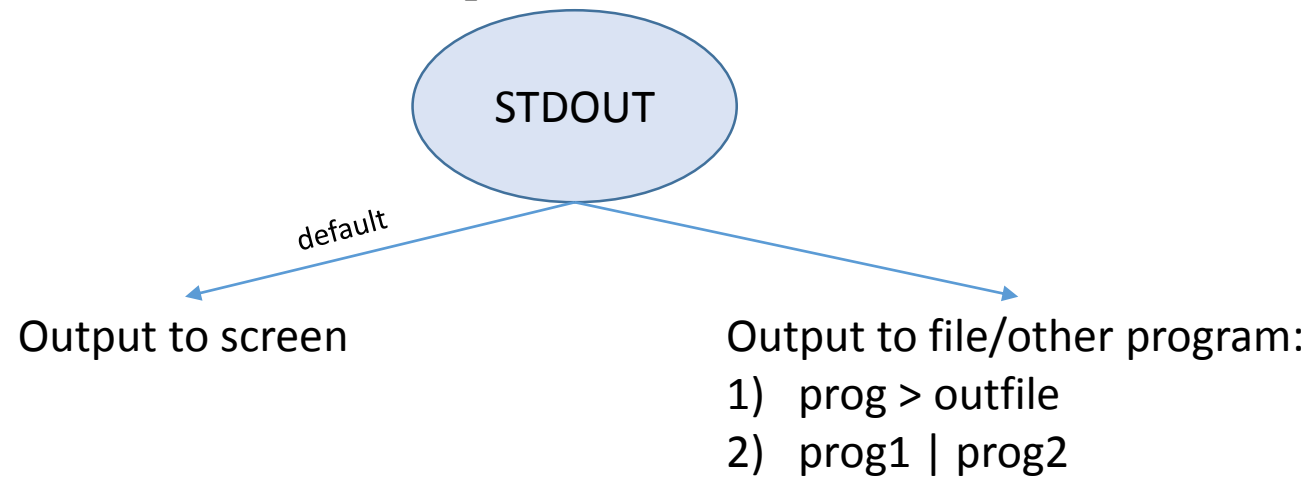
## Input Redirection



```
student@ubuntu:~/Desktop/I0$ ls -lrt
total 16
-rw-rw-r-- 1 student student 486 May 30 11:05 prog.c
-rw-rw-r-- 1 student student 23 May 30 11:07 f1
-rwxrwxr-x 1 student student 7263 May 30 11:11 prog
student@ubuntu:~/Desktop/I0$ ./prog
HELLO.
hello.
I AM A STUDENT.
i am a student.
student@ubuntu:~/Desktop/I0$
```

```
student@ubuntu:~/Desktop/I0$ ls -l
total 16
-rw-rw-r-- 1 student student 23 May 30 11:07 f1
-rwxrwxr-x 1 student student 7263 May 30 11:11 prog
-rw-rw-r-- 1 student student 486 May 30 11:05 prog.c
student@ubuntu:~/Desktop/I0$ cat f1
HELLO.
I AM A STUDENT.
student@ubuntu:~/Desktop/I0$ ./prog < f1
hello.
i am a student.
student@ubuntu:~/Desktop/I0$ cat f1 | ./prog
hello.
i am a student.
student@ubuntu:~/Desktop/I0$
```

## Output Redirection



```
student@ubuntu:~/Desktop/IO$ ls -lrt
total 16
-rw-rw-r-- 1 student student 486 May 30 11:05 prog.c
-rw-rw-r-- 1 student student 23 May 30 11:07 f1
-rwxrwxr-x 1 student student 7263 May 30 11:11 prog
student@ubuntu:~/Desktop/IO$ ./prog
HELLO.
hello.
I AM A STUDENT.
i am a student.
student@ubuntu:~/Desktop/IO$
```

```
student@ubuntu:~/Desktop/IO$ rm f1
student@ubuntu:~/Desktop/IO$ ls -l
total 12
-rwxrwxr-x 1 student student 7263 May 30 11:11 prog
-rw-rw-r-- 1 student student 486 May 30 11:05 prog.c
student@ubuntu:~/Desktop/IO$ ./prog > f1
HELLO.
I AM A STUDENT.
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student 23 May 30 11:44 f1
-rwxrwxr-x 1 student student 7263 May 30 11:11 prog
-rw-rw-r-- 1 student student 486 May 30 11:05 prog.c
student@ubuntu:~/Desktop/IO$ cat f1
hello.
i am a student.
student@ubuntu:~/Desktop/IO$ ./prog | cat
HELLO
I AM A STUDENT.
hello
i am a student.
student@ubuntu:~/Desktop/IO$
```

```
int printf(char *format, arg1, arg2, ...)  
int sprintf(char *string, char *format, arg1, arg2, ...)
```

The function **sprintf** does the same conversions as **printf** does, but stores the output in a string. **sprintf** formats the arguments in arg1, arg2, etc., according to format as printf do, but places the result in string instead of the standard output; string must be big enough to receive the result. As in **printf**, **sprintf** returns the number of characters printed.

```
#include <stdio.h>  
int main()  
{  
    char str[100];  
    int x =1, y=2, c;  
    c = sprintf(str, "Sum of %d + %d = %d\n", x, y, x+y);  
    printf("%d:%s\n",c, str);  
    return 0;  
}
```

**Output:**  
17:Sum of 1 + 2 = 3

Note:  
char\* str = "abc %d";  
Not safe: printf(str);  
Safe: printf("%s",str);



```
int scanf(char *format, arg1, arg2, ...)
```

```
int sscanf(char *string, char *format, arg1, arg2, ...)
```

- ✓ **scanf** reads characters from the standard input, interprets them according to the specification in format, and stores the results through the remaining arguments *each of which must be a pointer*, indicate where the corresponding converted input should be stored. **It returns the number of successfully matched and assigned input items**. The next call to scanf resumes searching immediately after the last character already converted.
- ✓ **sscanf** reads from a string instead of the standard input. It scans the string according to the format in format and stores the resulting values through arg1, arg2, etc.

```
#include <stdio.h>
int main()
{
    char* str = "31 May 2017";
    int day, year, c;
    char month[10];
    c = sscanf(str, "%d %s %d", &day, month, &year);
    printf("%d: Date is:%d %s %d\n", c, day, month, year);
    return 0;
}
```

**Output:**

3: Date is:31 May 2017

```
FILE *fp
FILE *fopen(char *name, char *mode)
int fclose(FILE *fp)
```

- ✓ **fp** is the *file pointer*, points to a structure that contains information about the file. (the location of a buffer, the current character position in the buffer, whether the file is being read or written, and whether errors or end of file have occurred).
- ✓ The FILE structure definitions can be obtained from <stdio.h>.
- ✓ char\* name: the name of the file.
- ✓ char\* mode: "r", "w" or "a" for read, write and append accordingly ("b" can be appended to the mode to specify it is binary file)
- ✓ If a file that does not exist is opened for writing or appending, it is created if possible.
- ✓ Opening an existing file for writing causes the old contents to be discarded while opening for appending preserves them.
- ✓ Trying to read a file that does not exist (or don't have permission) is an error.
- ✓ **If there is any error, fopen will return NULL.**
- ✓ **fclose**, it close the connection between the file pointer and the external name that was established by fopen.
- ✓ fclose on an output file - flushes the buffer in which putc is collecting output.
- ✓ fclose is called automatically for each open file when a program terminates normally.
- ✓ stdin and stdout can also be closed if they are not needed anymore.
- ✓ returns **zero** if file is successfully closed or **EOF** if failed.

mode	Description
"r"	Opens a file for reading. The file must exist.
"w"	Creates an empty file for writing. If a file with the same name already exists, its content is erased and the file is considered as a new empty file.
"a"	Appends to a file. Writing operations, append data at the end of the file. The file is created if it does not exist.
"r+"	Opens a file to update both reading and writing. The file must exist.
"w+"	Creates an empty file for both reading and writing.
"a+"	Opens a file for reading and appending.

```
int getc(FILE *fp)
int putc(int c, FILE *fp)
```

- ✓ **getc** returns the next character from the stream referred to by fp - it returns **EOF** for end of file or error.
- ✓ **putc** writes the character c to the file fp and **returns the character written**, or EOF if an error occurs

```
#include <stdio.h>
#include <ctype.h>
int main()
{
    int c;
    FILE *fp1,*fp2;
    fp1 = fopen("f1","r");
    fp2 = fopen("f2","w");

    while ((c = getc(fp1)) != EOF)
        putc(tolower(c),fp2);

    fclose(fp1);
    fclose(fp2);

    return 0;
}
```

prog.c

must check if files  
were successfully  
opened. How to  
do that? Check if  
fopen return  
NULL.

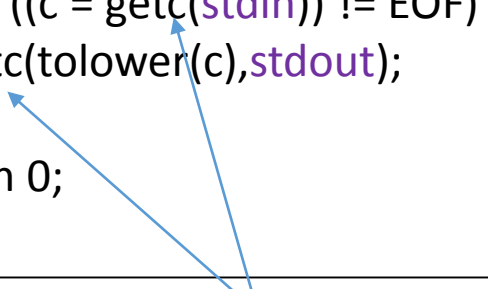
```
student@ubuntu:~/Desktop/I0$ ls -l
total 16
-rw-rw-r-- 1 student student 23 May 31 10:51 f1
-rwxrwxr-x 1 student student 7278 May 31 10:51 prog
-rw-rw-r-- 1 student student 219 May 31 10:51 prog.c
student@ubuntu:~/Desktop/I0$ cat f1
HELLO.
I AM A STUDENT.
student@ubuntu:~/Desktop/I0$ ./prog
student@ubuntu:~/Desktop/I0$ ls -l
total 20
-rw-rw-r-- 1 student student 23 May 31 10:51 f1
-rw-rw-r-- 1 student student 23 May 31 10:59 f2
-rwxrwxr-x 1 student student 7278 May 31 10:51 prog
-rw-rw-r-- 1 student student 219 May 31 10:51 prog.c
student@ubuntu:~/Desktop/I0$ cat f2
hello.
i am a student.
student@ubuntu:~/Desktop/I0$
```

```
#include <stdio.h>
#include <ctype.h>
int main()
{
    int c;

    while ((c = getc(stdin)) != EOF)
        putc(tolower(c), stdout);

    return 0;
}
```

prog.c



Exactly the same as getchar and putchar

```
student@ubuntu:~/Desktop/IO$ ./prog
HELLO.
hello.
I AM A STUDENT.
i am a student.
student@ubuntu:~/Desktop/IO$
```

```
int remove(char *name)
```

- ✓ removes the file name from the file system.
- ✓ name: name of file to remove
- ✓ returns 0 if success to remove the file and -1 if failed.

```
#include <stdio.h>
int main()
{
    int x,y;
    x = remove("f1");
    y = remove("f2");
    printf("%d %d\n",x, y);
    return 0;
}
```

```
student@ubuntu:~/Desktop/IO$ ls -l
total 12
-rw-rw-r-- 1 student student  0 Jun  1 10:30 f1
-rwxrwxr-x 1 student student 7198 Jun  1 10:30 prog
-rw-rw-r-- 1 student student 133 Jun  1 10:29 prog.c
student@ubuntu:~/Desktop/IO$ ./prog
0 -1
student@ubuntu:~/Desktop/IO$ ls -l
total 12
-rwxrwxr-x 1 student student 7198 Jun  1 10:30 prog
-rw-rw-r-- 1 student student 133 Jun  1 10:29 prog.c
student@ubuntu:~/Desktop/IO$
```

```
int fscanf(FILE *fp, char *format, ...)
int fprintf(FILE *fp, char *format, ...)
```

- ✓ fscanf and fprintf may be used for formatted input or output of files. identical to scanf and printf, except that the first argument is a file pointer that specifies the file to be read or written.
- ✓ fscanf returns the number of input items successfully matched and assigned
- ✓ fprintf If successful, the total number of characters written is returned otherwise, a negative number is returned.

```
#include <stdio.h>
int main()
{
    int day, year;
    char month[100];
    FILE *fp1, *fp2;
    fp1 = fopen("f1", "r");
    fp2 = fopen("f2", "w");

    fscanf(fp1, "%d %s %d", &day, month, &year);
    fprintf(fp2, "%d %s %d\n", day, month, year);

    fclose(fp1);  fclose(fp2);
    return 0;
}
```

prog.c

must check if files were successfully opened and if fscanf succeeded.

```
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student  12 Jun  4 02:54 f1
-rwxrwxr-x 1 student student 7322 Jun  4 02:55 prog
-rw-rw-r-- 1 student student  321 Jun  4 02:55 prog.c
student@ubuntu:~/Desktop/IO$ cat f1
31 May 2016
student@ubuntu:~/Desktop/IO$ ./prog
student@ubuntu:~/Desktop/IO$ ls -l
total 20
-rw-rw-r-- 1 student student  12 Jun  4 02:54 f1
-rw-rw-r-- 1 student student  12 Jun  4 02:56 f2
-rwxrwxr-x 1 student student 7322 Jun  4 02:55 prog
-rw-rw-r-- 1 student student  321 Jun  4 02:55 prog.c
student@ubuntu:~/Desktop/IO$ cat f2
31 May 2016
student@ubuntu:~/Desktop/IO$
```

## Shell Redirection of stdin, stdout and stderr.

Character	Action
>	Redirect standard output
2>	Redirect standard error
2>&1	Redirect standard error to standard output
<	Redirect standard input
	Pipe standard output to another command
>>	Append to standard output
2>&1	Pipe standard output and standard error to another command

```
#include <stdio.h>
#include <stdlib.h> /*exit ..*/
int main() {
    int x = 20, y = 0 , z;

    if( y == 0) {
        fprintf(stderr, "Division by zero! Exiting...\n");
        exit(EXIT_FAILURE);
    }
    z = x / y;
    return 0;
}
```

prog.c

The shell and many UNIX commands take their input from standard input (stdin), write output to standard output (stdout), and write error output to standard error (stderr). By default, standard input is connected to the terminal keyboard and standard output and error to the terminal screen.

```
student@ubuntu:~/Desktop/IO$ ls -l
total 12
-rwxrwxr-x 1 student student 7230 May 31 13:57 prog
-rw-rw-r-- 1 student student 225 May 31 13:57 prog.c
student@ubuntu:~/Desktop/IO$ ./prog
Division by zero! Exiting...
student@ubuntu:~/Desktop/IO$ ./prog 2> myErrFile
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student 29 May 31 13:57 myErrFile
-rwxrwxr-x 1 student student 7230 May 31 13:57 prog
-rw-rw-r-- 1 student student 225 May 31 13:57 prog.c
student@ubuntu:~/Desktop/IO$ cat myErrFile
Division by zero! Exiting...
student@ubuntu:~/Desktop/IO$
```

stdlib.h: defines several general purpose functions, including dynamic memory management, random number generation, communication with the environment, integer arithmetics, searching, sorting and converting.

```

int ferror(FILE *fp)
int feof(FILE *fp)
void perror(const char* str)
    stdin stdout stderr

```

**ferror** returns non-zero if an error occurred on the stream fp (generally set by a previous operation on the *stream* that failed).

**perror** print error message to stderr.

**feof**: it returns non-zero if end of file has occurred on the specified file.

```

#include <stdio.h>
int main ()
{
    FILE * pf = fopen("f1","r");
    if(pf==NULL)
        perror ("Error opening file");
    else
    {
        fputc ('A',pf);
        if (ferror (pf))
            printf ("Error:could'nt write to f1\n");
        fclose (pf);
    }
    return 0;
}

```

prog.c

Will cause failure as the file is opened for read only

```

student@ubuntu:~/Desktop/test$ ls -lrt
total 16
-rw-rw-r-- 1 student student 250 May 23 10:14 prog.c
-rwxrwxr-x 1 student student 7343 May 23 10:14 a.out
-rw-rw-r-- 1 student student 23 May 23 10:17 f1
student@ubuntu:~/Desktop/test$ cat f1
Hello.
I am a student.
student@ubuntu:~/Desktop/test$ ./a.out
Error:could'nt write to f1
student@ubuntu:~/Desktop/test$

```



```
char* gets(char* line)
int puts(char* line)
```

- ✓ gets: reads a line from stdin and stores it into the string pointed to by line. It stops when either the newline character is read or when the end-of-file is reached, whichever comes first.
  - **not safe** to use because it does not check the array bound.
- ✓ puts: writes a string to stdout up to but not including the null character. A newline character is appended to the output.
- ✓ gets: deletes the terminating '\n' however puts adds it.
- ✓ gets: returns line on success, and NULL on error or when end of file occurs, while no characters have been read.
- ✓ puts: If successful, non-negative value is returned. On error, the function returns EOF.

```
#include <stdio.h>
int main()
{
    char line[100];

    while(gets(line) != NULL)
    {
        puts(line);
    }

    return 0;
}
```

prog.c

```
student@ubuntu:~/Desktop/IO$ ./prog
Hello
Hello
I am a Student
I am a Student
student@ubuntu:~/Desktop/IO$
```

```
char *fgets(char *line, int maxline, FILE *fp)
int fputs(char *line, FILE *fp)
```

- ✓ **fgets** reads the next input line (including the newline) from file fp into the character array line
  - maxline: at most maxline-1 characters will be read.
  - The resulting line is terminated with '\0'.
  - It stops when either (n-1) characters are read, the newline character is read, or the end-of-file is reached, whichever comes first.
  - **safe** to use because it checks the array bound.
  - Normally fgets returns line; on end of file or error it returns NULL.
- ✓ **fputs** writes a string (which need not contain a newline) to a file.
  - It returns EOF if an error occurs, and non-negative otherwise.

```
#include <stdio.h>
int main() {
    char line[100];
    FILE *fp1, *fp2;
    fp1 = fopen("f1","r");
    fp2 = fopen("f2","w");

    while(fgets(line, 100, fp1) != NULL)
    {
        fputs(line, fp2);
    }

    return 0;
}
```

prog.c

```
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student  23 May 31 16:21 f1
-rwxrwxr-x 1 student student 7281 May 31 16:20 prog
-rw-rw-r-- 1 student student  244 May 31 16:20 prog.c
student@ubuntu:~/Desktop/IO$ cat f1
Hello.
I am a student.
student@ubuntu:~/Desktop/IO$ ./prog
student@ubuntu:~/Desktop/IO$ ls -l
total 20
-rw-rw-r-- 1 student student  23 May 31 16:21 f1
-rw-rw-r-- 1 student student  23 May 31 16:21 f2
-rwxrwxr-x 1 student student 7281 May 31 16:20 prog
-rw-rw-r-- 1 student student  244 May 31 16:20 prog.c
student@ubuntu:~/Desktop/IO$ cat f2
Hello.
I am a student.
student@ubuntu:~/Desktop/IO$
```

## Random Access

```
int fseek(FILE *stream, long int offset, int whence)
```

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int n=0;
    char* fname = "/home/student/Desktop/IO/f1";
    FILE* fp = fopen(fname,"r+");
    if(fp == NULL) {
        fprintf(stderr,"failed to open file %s\n",fname);
        exit(1);
    }
    n = fseek(fp,0,2);
    if(n != 0) {
        fprintf(stderr,"failed to move cursor of file %s\n",fname); exit(2);
    }
    if(fputs("3 My name is Alex\n",fp) == EOF) {
        fprintf(stderr,"failed to write to file %s\n",fname); exit(3);
    }
    fclose(fp); return 0;
}
```

- ✓ **offset** – This is the number of bytes to offset from whence.
- ✓ **whence** – This is the position from where offset is added. It is specified by one of the following constants : 0, 1 or 2 (SEEK\_SET , SEEK\_CUR or SEEK\_END,...)
- ✓ returns zero if successful others non-zero value is returned.

```
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student  23 Jun  2 10:45 f1
-rwxrwxr-x 1 student student 7380 Jun  2 10:46 prog
-rw-rw-r-- 1 student student  620 Jun  2 10:46 prog.c
student@ubuntu:~/Desktop/IO$ cat f1
Hello.
I am a student.
student@ubuntu:~/Desktop/IO$ ./prog
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student  41 Jun  2 10:46 f1
-rwxrwxr-x 1 student student 7380 Jun  2 10:46 prog
-rw-rw-r-- 1 student student  620 Jun  2 10:46 prog.c
student@ubuntu:~/Desktop/IO$ cat f1
Hello.
I am a student.
3 My name is Alex
student@ubuntu:~/Desktop/IO$
```

## Command Execution

### system(char \*s)

The function system(char \*s) executes the command contained in the character string s, then resumes execution of the current program.

```
#include <stdlib.h>
int main()
{
    system("mkdir MyDir");

    return 0;
}
```

prog.c

```
student@ubuntu:~/Desktop/IO$ ls -l
total 12
-rwxrwxr-x 1 student student 7160 May 31 17:07 prog
-rw-rw-r-- 1 student student  78 May 31 17:06 prog.c
student@ubuntu:~/Desktop/IO$ ./prog
student@ubuntu:~/Desktop/IO$ ls -l
total 16
drwxrwxr-x 2 student student 4096 May 31 17:07 MyDir
-rwxrwxr-x 1 student student 7160 May 31 17:07 prog
-rw-rw-r-- 1 student student  78 May 31 17:06 prog.c
student@ubuntu:~/Desktop/IO$
```

## Low Level I/O - Read and Write

```
int read(int fd, char *buf, int n);  
int write(int fd, char *buf, int n);
```

- ✓ fd: a file descriptor (**0,1 or 2**).
- ✓ buf: a character array where the data is to go to or to come from.
- ✓ n: number of bytes to be transferred (1,1024,4096.. larger sizes will be more efficient because fewer system calls will be made).
- ✓ Each returns a count of the number of bytes transferred. On reading, the number of bytes returned may be less than the number requested. A return value of **zero bytes implies end of file**, and **-1 indicates an error** of some sort. For writing, the return value is the number of bytes written; an error has occurred if this isn't equal to the number requested.

```
#include <stdio.h>  
#include <unistd.h> /*read,write,close*/  
#define BUFSIZE 4096  
int main()  
{  
    char buf[BUFSIZE];  
    int n;  
    while ((n = read(0, buf, BUFSIZE)) > 0)  
    {  
        write(1, buf, n);  
    }  
    return 0;  
}
```

unistd.h: provides access to the POSIX operating system API.  
On Unix-like systems, the interface defined by unistd.h is typically made up largely of system call wrapper functions such as fork, pipe and I/O primitives (read, write, close, etc.  
POSIX: **Portable Operating System Interface (POSIX)** is a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. POSIX defines the application programming interface (API), along with command line shells and utility interfaces, for software compatibility with variants of Unix and other operating systems

```
int open(char *name, int flags, int perms)
int creat(char *name, int perms)
close(char* name)
```

- ✓ open/create return a file descriptor (int value).
- ✓ open returns -1 if any error occurs (open return error if file does not exist).
- ✓ create returns a file descriptor if success, and -1 if not (file will be truncated to zero length if exists, thereby discarding its previous contents)
- ✓ name: the filename
- ✓ flags: specifies how the file is to be opened (O\_RDONLY , O\_WRONLY , O\_RDWR, - These constants are defined in <fcntl.h>)
- ✓ perms : 3 octal numbers define the permission (rwx) for the owner/group/others e.g 0666 specifies RW for owner, group and others.
- ✓ Open always use 0.

```
#include <stdio.h>
#include <fcntl.h> /*open,create, O_RDONLY*/
#include <unistd.h> /*read,write,close*/
int main() {
    int f1, f2, n;
    char buf[BUFSIZ];
    f1 = open("/home/student/Desktop/IO/f1", O_RDONLY, 0);
    f2 = creat("/home/student/Desktop/IO/f2", 0666);

    if (f1 == -1)    printf("Failed open file f1\n");
    if (f2 == -1)    printf("Failed creating file f2\n");

    while ((n = read(f1, buf, BUFSIZ)) > 0)
        if (write(f2, buf, n) != n)
            printf("Error while write to file\n");
    close(f1); close(f2);
    return 0;
}
```

Copy content of file f1 to f2.  
Create f2 if dose not exists

```
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student  23 Jun  1 09:48 f1
-rwxrwxr-x 1 student student 7352 Jun  1 10:04 prog
-rw-rw-r-- 1 student student  503 Jun  1 10:01 prog.c
student@ubuntu:~/Desktop/IO$ cat f1
Hello.
I am a student.
student@ubuntu:~/Desktop/IO$ ./prog
student@ubuntu:~/Desktop/IO$ ls -l
total 20
-rw-rw-r-- 1 student student  23 Jun  1 09:48 f1
-rw-rw-r-- 1 student student  23 Jun  1 10:06 f2
-rwxrwxr-x 1 student student 7352 Jun  1 10:04 prog
-rw-rw-r-- 1 student student  503 Jun  1 10:01 prog.c
student@ubuntu:~/Desktop/IO$ cat f2
Hello.
I am a student.
student@ubuntu:~/Desktop/IO$ pwd
/home/student/Desktop/IO
student@ubuntu:~/Desktop/IO$
```

fcntl.h: header in the C POSIX library for the C programming language that contains constructs that refer to file control, e.g. opening a file, retrieving and changing the permissions of file, locking a file for edit, etc

```
int unlink(char *name)
```

- ✓ Similar to the 'standard library function remove' , unlike removes the file name from the file system.
- ✓ name: name of file to remove
- ✓ returns 0 if success to remove the file and -1 if failed.

```
#include <stdio.h>
#include <unistd.h>
int main()
{
    int x, y;
    x = unlink("f1");
    y = unlink("f2");
    printf("%d %d\n",x,y);
    return 0;
}
```

```
student@ubuntu:~/Desktop/I0$ ls -l
total 12
-rw-rw-r-- 1 student student  0 Jun  1 10:26 f1
-rwxrwxr-x 1 student student 7198 Jun  1 10:26 prog
-rw-rw-r-- 1 student student 133 Jun  1 10:26 prog.c
student@ubuntu:~/Desktop/I0$ ./prog
0 -1
student@ubuntu:~/Desktop/I0$ ls -l
total 12
-rwxrwxr-x 1 student student 7198 Jun  1 10:26 prog
-rw-rw-r-- 1 student student 133 Jun  1 10:26 prog.c
student@ubuntu:~/Desktop/I0$
```

## Random Access

```
long lseek(int fd, long offset, int origin)
```

- ✓ each read or write takes place at a position in the file right after the previous one.
- ✓ the system call lseek provides a way to move around in a file without reading or writing any data.
- ✓ sets the current position in the file to offset, which is taken relative to the location specified by origin (0, 1, or 2: beginning, current position or end of file respectively) e.g. to append to a file (the redirection >> in the UNIX shell, or "a" for fopen), seek to the end before writing → lseek(fd, 0L, 2). To set it back to the beginning of the file → lseek(fd, 0L, 0).
- ✓ lseek return the new position in the file, or -1 if an error occurs.

```
#include <stdio.h>
#include <unistd.h> /* read,write..*/
#include <fcntl.h> /* O_RDONLY*/
#include <stdlib.h> /*exit()*/
#include <string.h> /*strlen*/
int main(){
    char* fname = "/home/student/Desktop/IO/f1";
    int n=0, fd = open(fname, O_WRONLY, 0);
    if(fd == -1){
        fprintf(stderr,"failed to open file %s\n",fname);
        exit(1);
    }
    n = lseek(fd,0,2);
    if(n == -1){
        fprintf(stderr,"failed to move cursor of file %s\n",fname); exit(2);
    }
    n = write(fd,"3 My name is Alex\n",strlen("3 My name is Alex\n"));
    if(n == -1){
        fprintf(stderr,"failed to write to file %s\n",fname);    exit(3);
    }
    close(fd); return 0;
}
```

```
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student  23 Jun  2 10:06 f1
-rwxrwxr-x 1 student student 7378 Jun  2 10:05 prog
-rw-rw-r-- 1 student student  661 Jun  2 10:04 prog.c
student@ubuntu:~/Desktop/IO$ cat f1
Hello.
I am a student.
student@ubuntu:~/Desktop/IO$ ./prog
student@ubuntu:~/Desktop/IO$ ls -l
total 16
-rw-rw-r-- 1 student student  41 Jun  2 10:06 f1
-rwxrwxr-x 1 student student 7378 Jun  2 10:05 prog
-rw-rw-r-- 1 student student  661 Jun  2 10:04 prog.c
student@ubuntu:~/Desktop/IO$ cat f1
Hello.
I am a student.
3 My name is Alex
student@ubuntu:~/Desktop/IO$
```



## File Information

```
int stat(char *fname, struct stat *stBufp)
```

- ✓ fname: file name
- ✓ stBufp: pointer to a structure of type stat.
- ✓ stat takes a filename and fills the structure pointed by stBufp with the inode information for that file.
- ✓ returns -1 if there is an error and zero in success

```
#include <stdio.h>
#include <sys/stat.h>
#include <time.h>
int main(void)
{
    char *name = "f1", str[25];
    struct stat stbuf;
    stat(name, &stbuf);
    strftime(str, 25, "%d.%m.%Y %H:%M:%S", localtime(&stbuf.st_ctime));
    printf("Inode:%ld Size:%ld Creation Time:%s\n", stbuf.st_ino, stbuf.st_size, str);

    return 0;
}
```

prog.c

```
student@ubuntu:~/Desktop/I0$ ls -l
total 16
-rw-rw-r-- 1 student student 23 Jun  3 10:09 f1
-rwxrwxr-x 1 student student 7372 Jun  3 10:35 prog
-rw-rw-r-- 1 student student 354 Jun  3 10:34 prog.c
student@ubuntu:~/Desktop/I0$ ./prog
Inode:1300876 Size:23 Creation Time:03.06.2017 10:09:02
student@ubuntu:~/Desktop/I0$
```

The structure `stat` is declared in `<sys/stat.h>`, and looks like:  
`struct stat /* inode information returned by stat */`

```
{
    dev_t st_dev; /* device of inode */
    ino_t st_ino; /* inode number */
    short st_mode; /* mode bits */
    short st_nlink; /* number of links to file */
    short st_uid; /* owners user id */
    short st_gid; /* owners group id */
    dev_t st_rdev; /* for special files */
    off_t st_size; /* file size in characters */
    time_t st_atime; /* time last accessed */
    time_t st_mtime; /* time last modified */
    time_t st_ctime; /* time originally created */
};
```

The `st_mode` entry contains a set of flags describing the file. The flag definitions are also included in `<sys/types.h>`; we need only the part that deals with file type:

```
#define S_IFMT 0160000 /* type of file: */
#define S_IFDIR 0040000 /* directory */
#define S_IFCHR 0020000 /* character special */
#define S_IFBLK 0060000 /* block special */
#define S_IFREG 0010000 /* regular */
```

```
#define _BSD_SOURCE
```

```
#include <stdio.h>
```

```
#include <sys/stat.h>
```

```
int main(void) {
```

```
    char *name = "f1";
```

```
    struct stat stbuf;
```

```
    stat(name, &stbuf);
```

```
    switch (stbuf.st_mode & S_IFMT) {
```

```
    case S_IFREG:
```

```
        puts("regular file");
```

```
        break;
```

```
    case S_IFDIR:
```

```
        puts("directory");
```

```
        break;
```

```
    case S_IFCHR:
```

```
        puts("character device");
```

```
        break;
```

```
    case S_IFBLK:
```

```
        puts("block device");
```

```
        break;
```

```
    case S_IFLNK:
```

```
        puts("symbolic link");
```

```
        break;
```

```
    case S_IFIFO:
```

```
        puts("pipe");
```

```
        break;
```

```
    case S_IFSOCK:
```

```
        puts("socket");
```

```
        break;
```

```
    default:
```

```
        puts("unknown");
```

```
    }
```

```
    return 0;
```

```
}
```

prog.c

A program to check the type of a file

```
student@ubuntu:~/Desktop/IO$ ls -l
total 20
drwxrwxr-x 2 student student 4096 Jun  3 11:34 A
-rw-rw-r-- 1 student student  23 Jun  3 10:09 f1
-rwxrwxr-x 1 student student 7241 Jun  3 11:42 prog
-rw-rw-r-- 1 student student  831 Jun  3 11:41 prog.c
student@ubuntu:~/Desktop/IO$ ./prog
directory
student@ubuntu:~/Desktop/IO$
```

```
#define _BSD_SOURCE
```

above define has to be added to the program

others compilation may fail why?

Is this compilation issue related to ANSI flag?

prog.c

```
#include <stdio.h>
#include <dirent.h>
int main(void)
{
    DIR *d=NULL;
    struct dirent *dir=NULL;
    d = opendir(".");
    if (d)
    {
        while ((dir = readdir(d)) != NULL)
        {
            printf("%s\n", dir->d_name);
        }
        closedir(d);
    }
    return(0);
}
```

For additional information please see  
chapter 8 From the book **(a must)**  
8.6 Example - Listing Directories

show content of directory

```
student@ubuntu:~/Desktop/IO$ ls -l
total 20
drwxrwxr-x 2 student student 4096 Jun  3 11:34 A
-rw-rw-r-- 1 student student  23 Jun  3 10:09 f1
-rwxrwxr-x 1 student student 7276 Jun  3 12:05 prog
-rw-rw-r-- 1 student student  280 Jun  3 12:05 prog.c
student@ubuntu:~/Desktop/IO$ ./prog
A
.
..
prog.c
prog
f1
.
student@ubuntu:~/Desktop/IO$
```

The dirent structure is declared as follows:

```
struct dirent {
    long d_ino;                /* inode number */
    off_t d_off;               /* offset to this dirent */
    unsigned short d_reclen;    /* length of this d_name */
    char d_name [NAME_MAX+1];  /* filename (null-terminated) */
}
```



***END***