

# Standard Library in C

# Outline

- ❖ Introduction
- ❖ `stdio.h`
- ❖ `stdlib.h`
- ❖ `ctype.h`
- ❖ `stdarg.h`
- ❖ `math.h`
- ❖ `string.h`
- ❖ `assert.h`
- ❖ `errno.h`
- ❖ `time.h`

# Introduction

## ❖ Standard library:

- type definitions
- variable declarations
- constant and macro definitions
- functions

## ❖ Description and usage information can be obtained from man pages on unix-like OS or the web

- section 3
- on unix and unix-like OS: in the terminal type: `man [<section>] <library_function_name>`
- Many websites host copies of the man pages: [Die](#), [HE](#), [MAN7](#), ...

## ❖ List of standard library header files:

assert.h          ctype.h    errno.h    float.h    limits.h    locale.h    math.h  
                  setjmp.h  
signal.h          stdarg.h    [stddef.h](#)    stdio.h    stdlib.h    string.h    time.h

# Library `stdio.h`

## ❖ Types

- `size_t`
- `FILE`

## ❖ Constants

- `NULL`
- `EOF`
- `SEEK_CUR`  
`SEEK_END`  
`SEEK_SET`
- `stderr`  
`stdin`  
`stdout`

## ❖ Functions

- `FILE *fopen(const char *, const char *)`
- `int fclose(FILE *)`
- `int fflush(FILE *)` -----  
-----
- `int getchar(void)`
- `char *gets(char *)`
- `int scanf(const char *, ...)`
- `int putchar(int char)`
- `int puts(const char *)`
- `int printf(const char *, ...)` -----  
-----
- `int fgetc(FILE *)`
- `int ungetc(int char, FILE *stream)`
- `char *fgets(char *, int , FILE *)`
- `int fscanf(FILE *, const char *, ...)`
- `int fputc(int, FILE *)`
- `int fputs(const char *, FILE *)`
- `int fprintf(FILE *, const char *, ...)` -----  
-----

# Library `stdio.h`

## ❖ Functions (cont.)

- o `size_t fread(void *, size_t, size_t, FILE *)`
- o `size_t fwrite(const void *, size_t, size_t, FILE *)`
- 
- o `void rewind(FILE *)`
- o `int fseek(FILE *, long int, int)`
- o `int fgetpos(FILE *, fpos_t *)`
- o `int fsetpos(FILE *, const fpos_t *)`
- o `long int ftell(FILE *)`
- o -----
- o `int remove(const char *)`
- o `int rename(const char *, const char *)`
-

# Library `stdlib.h`

## ❖ Types

- `size_t`

## ❖ Constants

- `NULL`
- `EXIT_FAILURE`  
`EXIT_SUCCESS`
- `RAND_MAX`

## ❖ Functions (cont.)

- `void *malloc(size_t)`
- `void *calloc(size_t, size_t)`
- `void *realloc(void *, size_t)`
- `void free(void *)`  
-----
- `double atof(const char *)`
- `int atoi(const char *)`
- `long int atol(const char *)`
- `double strtod(const char *, char **)`
- `long int strtol(const char *, char **, int)`
- `unsigned long int strtoul(const char *, char **, int)`  
-----
- `void abort(void)`
- `void exit(int)`
- `int atexit(void (*func)(void))`
- `int system(const char *string)`  
-----
- `int abs(int x)`
- `long int labs(long int x)`

# Library `stdlib.h`

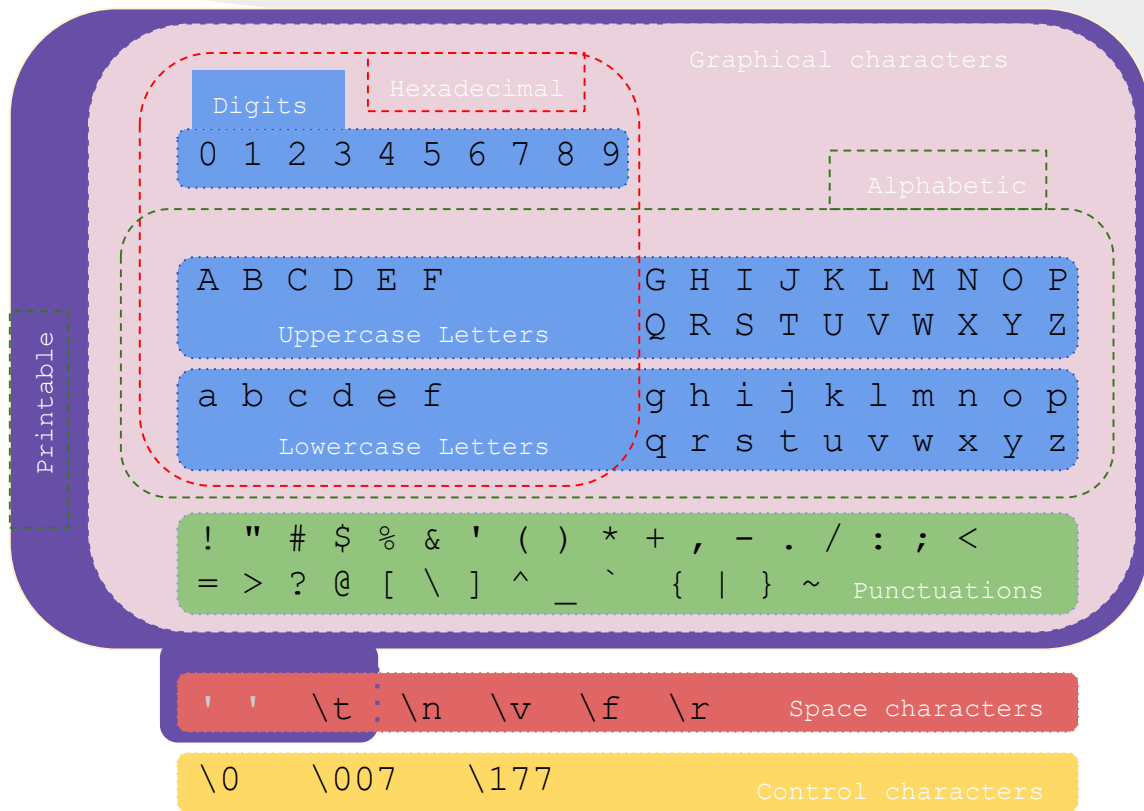
## ❖ Functions (cont.)

- o `int rand(void)`
- o `void srand(unsigned int seed)`  
-----
- o `void *bsearch(const void *, const void *, size_t, size_t,  
int (*compar)(const void *, const void *))`
- o `void qsort(void *, size_t, size_t, int (*compar)(const void *, const void*))`

# Library `ctype.h`

## ❖ Functions (cont.)

- o `int isalnum(int c)`
  - o `int isalpha(int c)`
  - o `int iscntrl(int c)`
  - o `int isdigit(int c)`
  - o `int isgraph(int c)`
  - o `int islower(int c)`
  - o `int isprint(int c)`
  - o `int ispunct(int c)`
  - o `int isspace(int c)`
  - o `int isupper(int c)`
  - o `int isxdigit(int c)`
- 
- o `int tolower(int c)`
  - o `int toupper(int c)`





# Library `stdarg.h`

## ❖ Types

- `va_list`

## ❖ Macros

- `void va_start(va_list, last_arg)`
- `type va_arg(va_list, type)`
- `void va_end(va_list)`
- `void va_copy(va_list, va_list)`

# Optional Parameters

- ❑ C permits functions to have optional parameters
- ❑ Syntax:           <returntype> <name>(<paramslist>, ...)
  - ... indicates that further parameters can be passed, must be listed only after the required parameters
  - since you specify the parameters as ..., you do not know their names!
- ❑ How to use these additional parameters when they are passed?
  - `stdarg.h` file contains the definition of `va_list` (variable argument list)
  - declare a variable of type `va_list`
  - use the macro `va_start` which initializes your variable to the first of the optional params
  - use the function `va_arg` which returns the next argument

# Optional Parameters

## ❑ Example:

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

int sum(int, ...);

int main(){
    printf("Sum of 15 and 56 = %d\n",    sum(2, 15, 56) );
    return 0;
}

int sum(int num_args, ...){
    int val = 0;
    va_list ap;
    int i;
    va_start(ap, num_args);
    for(i = 0; i < num_args; i++)
        val += va_arg(ap, int);
    va_end(ap);
    return val;
}
```

# Library `math.h`

## ❖ Arithmetic functions

- `double fabs(double )`  $|x|$
- `double ceil(double )`  $\lceil x \rceil$
- `double floor(double )`  $\lfloor x \rfloor$
- `double fmod(double , double )`  $x \% y$
- `double modf(double , double *)`  $x - \lfloor x \rfloor$  ,  $\lfloor x \rfloor$

## ❖ Exponential functions

- `double pow(double , double )`  $x^y$
- `double sqrt(double )`  $\sqrt{x}$
- `double exp(double )`  $e^x$
- `double ldexp(double , int )`  $x \cdot 2^y$
- `double log(double )`  $\log_e x$
- `double log10(double )`  $\log_{10} x$

## ❖ Trigonometric functions

- `double sin(double )`  $\sin(x)$
- `double cos(double )`  $\cos(x)$
- `double asin(double )`  $\sin^{-1}(x)$
- `double acos(double )`  $\cos^{-1}(x)$
- `double atan(double )`  $\tan^{-1}(x)$

- ❑ All functions take and yields double precision floating point values.
- ❑ Trigonometric functions deals with input and output angles in radians.

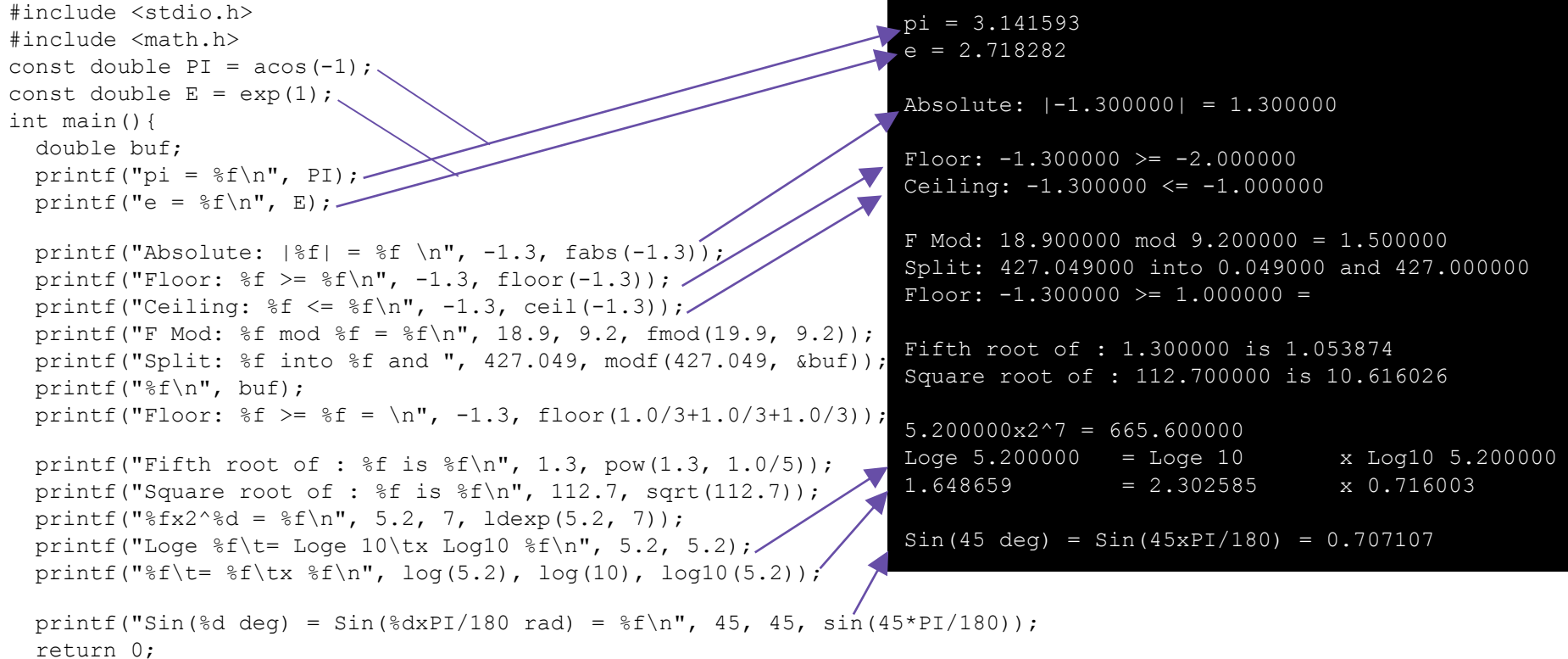
# Example

```
#include <stdio.h>
#include <math.h>
const double PI = acos(-1);
const double E = exp(1);
int main(){
    double buf;
    printf("pi = %f\n", PI);
    printf("e = %f\n", E);

    printf("Absolute: |%f| = %f \n", -1.3, fabs(-1.3));
    printf("Floor: %f >= %f\n", -1.3, floor(-1.3));
    printf("Ceiling: %f <= %f\n", -1.3, ceil(-1.3));
    printf("F Mod: %f mod %f = %f\n", 18.9, 9.2, fmod(18.9, 9.2));
    printf("Split: %f into %f and ", 427.049, modf(427.049, &buf));
    printf("%f\n", buf);
    printf("Floor: %f >= %f = \n", -1.3, floor(1.0/3+1.0/3+1.0/3));

    printf("Fifth root of : %f is %f\n", 1.3, pow(1.3, 1.0/5));
    printf("Square root of : %f is %f\n", 112.7, sqrt(112.7));
    printf("%fx2^%d = %f\n", 5.2, 7, ldexp(5.2, 7));
    printf("Loge %f\t= Loge 10\t\tLog10 %f\n", 5.2, 5.2);
    printf("%f\t= %f\t\tLog10 %f\n", log(5.2), log(10), log10(5.2));

    printf("Sin(%d deg) = Sin(%dxPI/180 rad) = %f\n", 45, 45, sin(45*PI/180));
    return 0;
}
```



The diagram illustrates the mapping between specific lines of C code and their corresponding output results. Purple arrows originate from the code on the left and point to the output on the right. The arrows show the following connections:

- From `printf("pi = %f\n", PI);` to `pi = 3.141593`
- From `printf("e = %f\n", E);` to `e = 2.718282`
- From `printf("Absolute: |%f| = %f \n", -1.3, fabs(-1.3));` to `Absolute: |-1.300000| = 1.300000`
- From `printf("Floor: %f >= %f\n", -1.3, floor(-1.3));` to `Floor: -1.300000 >= -2.000000`
- From `printf("Ceiling: %f <= %f\n", -1.3, ceil(-1.3));` to `Ceiling: -1.300000 <= -1.000000`
- From `printf("F Mod: %f mod %f = %f\n", 18.9, 9.2, fmod(18.9, 9.2));` to `F Mod: 18.900000 mod 9.200000 = 1.500000`
- From `printf("Split: %f into %f and ", 427.049, modf(427.049, &buf));` to `Split: 427.049000 into 0.049000 and 427.000000`
- From `printf("%f\n", buf);` to `Floor: -1.300000 >= 1.000000 =`
- From `printf("Fifth root of : %f is %f\n", 1.3, pow(1.3, 1.0/5));` to `Fifth root of : 1.300000 is 1.053874`
- From `printf("Square root of : %f is %f\n", 112.7, sqrt(112.7));` to `Square root of : 112.700000 is 10.616026`
- From `printf("%fx2^%d = %f\n", 5.2, 7, ldexp(5.2, 7));` to `5.200000x2^7 = 665.600000`
- From `printf("Loge %f\t= Loge 10\t\tLog10 %f\n", 5.2, 5.2);` to the first part of the log output: `Loge 5.200000 = Loge 10`
- From `printf("%f\t= %f\t\tLog10 %f\n", log(5.2), log(10), log10(5.2));` to the second part of the log output: `1.648659 = 2.302585`
- From `printf("Sin(%d deg) = Sin(%dxPI/180 rad) = %f\n", 45, 45, sin(45*PI/180));` to `Sin(45 deg) = Sin(45xPI/180) = 0.707107`

The output results are displayed on a black background with white text, while the code is on a white background with black text.

# Library `string.h`

## ❖ Memory functions

- o `int memcmp(const void *, const void *, size_t)`
- o `void *memchr(const void *, int, size_t)`
- o `void *memcpy(void *, const void *, size_t)`
- o `void *memset(void *, int, size_t )`

## ❖ String functions

- o `size_t strlen(const char *)`
- o `char *strcat(char *, const char *)`
- o `char *strncat(char *, const char *, size_t )`
- o `char *strcpy(char *, const char *)`
- o `char *strncpy(char *, const char *, size_t )`
- o `int strcmp(const char *, const char *)`
- o `int strncmp(const char *, const char *, size_t )`
- o `char *strchr(const char *, int)`
- o `char *strrchr(const char *, int)`
- o `char *strstr(const char *, const char *)`
- o `char *strpbrk(const char *, const char *)`
- o `size_t strspn(const char *, const char *)`
- o `size_t strcspn(const char *, const char *)`
- o `char *strtok(char *, const char *)`

- ❑ In coping functions, the first parameter is the destination and the second is the source.
- ❑ In search functions, first parameter is the haystack (text) and the second is the needle (pattern).

# Example

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

int main(){
    int* pArr = (int*)malloc(10*sizeof(int));
    char sentence[255], *word;
    memset(pArr, -100, 10*sizeof(int));
    printf("pArr[8]=%d\n", (char)pArr[8]);
    int iArr[] = {-3, 5, 0, 12, -8, 27};
    memcpy(pArr, iArr, 6*sizeof(int));
    printf("The 2 arrays are%s equal\n",memcmp(pArr,iArr,6*sizeof(int))?" not":"");
    int* ind = (int*)memchr(pArr, 12, 6*sizeof(int));
    printf("%d exist at index %d\n", 12, (int)(ind-pArr));

    char* name = "Adam";
    printf("Length of string %s is %d\n", name, (int)strlen(name));
    sprintf(sentence, "Length of string %s is %d\n", name, (int)strlen(name));
    word = strtok(sentence, " ");
    do {
        printf("%s\n", word);
    } while (word = strtok(NULL, " "));

    return 0;
}
```

```
pArr[8]=-100
The 2 arrays are equal
12 exist at index 3
Length of string Adam is 4
Length
of
string
Adam
is
4
```

# Libraries: `assert.h`, `errno.h` and `time.h`

## ❖ Macro of `assert.h`

- `void assert(int expression)`

## ❖ Macro of `errno.h`

- `extern int errno`

## ❖ `time.h`

- `clock_t`
- `time_t`
- `struct tm`

## ❖ Functions of `time.h`

- `clock_t clock()`
- `time_t time(time_t*)`
- `double difftime(time_t, time_t)`
- `time_t mktime(struct tm*)`
- `char* asctime(const struct tm*)`
- `char* ctime(const time_t)`
- `struct tm* gmtime(const time_t)`
- `struct tm* localtime(const time_t)`
- `size_t strftime(char* , size_t , const char* , const struct tm* )`

```
struct tm {  
    int tm_sec;      /* Seconds (0-60) */  
    int tm_min;      /* Minutes (0-59) */  
    int tm_hour;     /* Hours (0-23) */  
    int tm_mday;     /* Day of the month (1-31) */  
    int tm_mon;      /* Month (0-11) */  
    int tm_year;     /* Year - 1900 */  
    int tm_wday;     /* Day of week (0-6, Sunday=0) */  
    int tm_yday;     /* Day in year (0-365, 1 Jan=0) */  
    int tm_isdst;    /* Daylight saving time */  
};
```

Unix time epoch:  
1970, Jan, 1 00:00:00 UTC



# Example

```
#include <stdio.h>
#include <time.h>
int main(){
    time_t rawtime;
    struct tm * timeinfo;
    char buffer [80];

    time(&rawtime);
    printf("%s\n", ctime(&rawtime));

    timeinfo = localtime(&rawtime);
    printf("%s\n", asctime(timeinfo));

    strftime(buffer,80,"Now it's %y/%m/%d.",timeinfo);
    puts(buffer);
    strftime(buffer,80,"Now it's %Y/%m/%d.",timeinfo);
    puts(buffer);
    return 0;
}
```

```
Tue Apr 18 04:55:50 2017
Tue Apr 18 04:55:50 2017
Now it's 17/04/18.
Now it's 2017/04/18.
```

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

int main(){
    clock_t start_t, end_t;
    float total_t;
    int i;
    start_t = clock();
    printf("Starting @ start_t = %ld\n", start_t);
    printf("Run a big loop\n", start_t);
    for(i=0; i< 10000000; i++) { }
    end_t = clock();
    printf("Ending @ end_t = %ld\n", end_t);
    total_t=1000*(float) (end_t-start_t)/CLOCKS_PER_SEC;
    printf("Total CPU time: %f ms\n",total_t );
    return(0);
}
```

```
Starting @ start_t = 7865
Run a big loop
Ending @ end_t = 7915
Total CPU time: 0.050000 ms
```