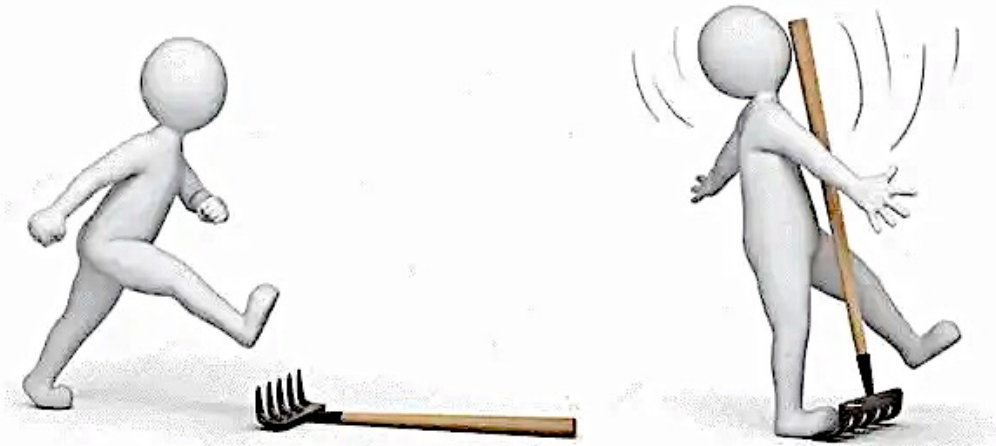
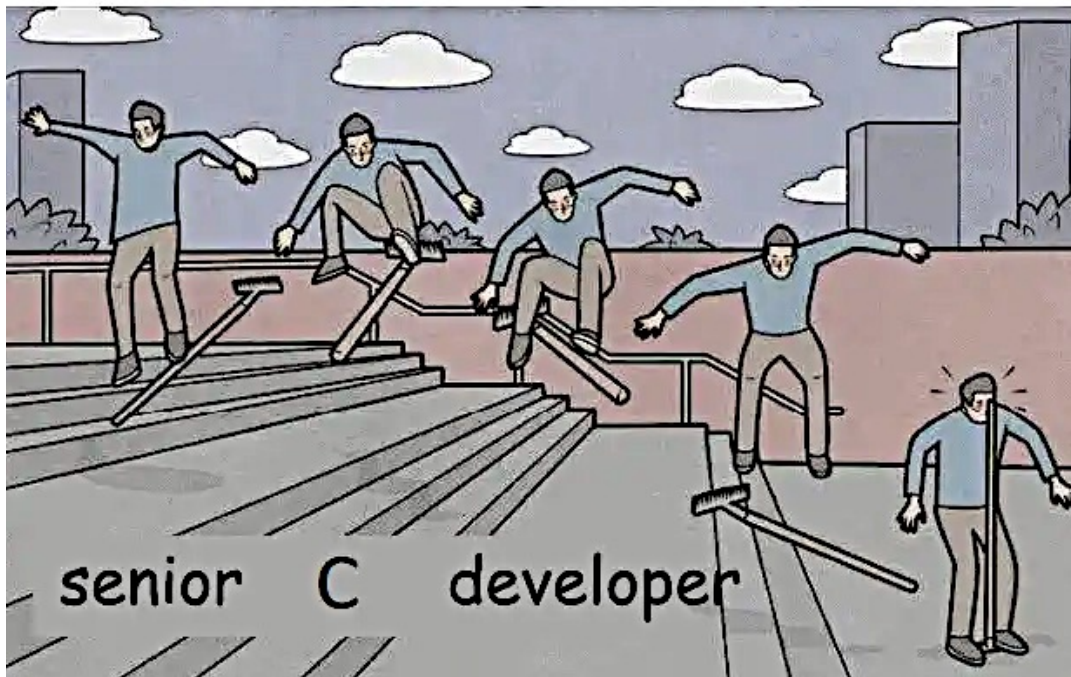
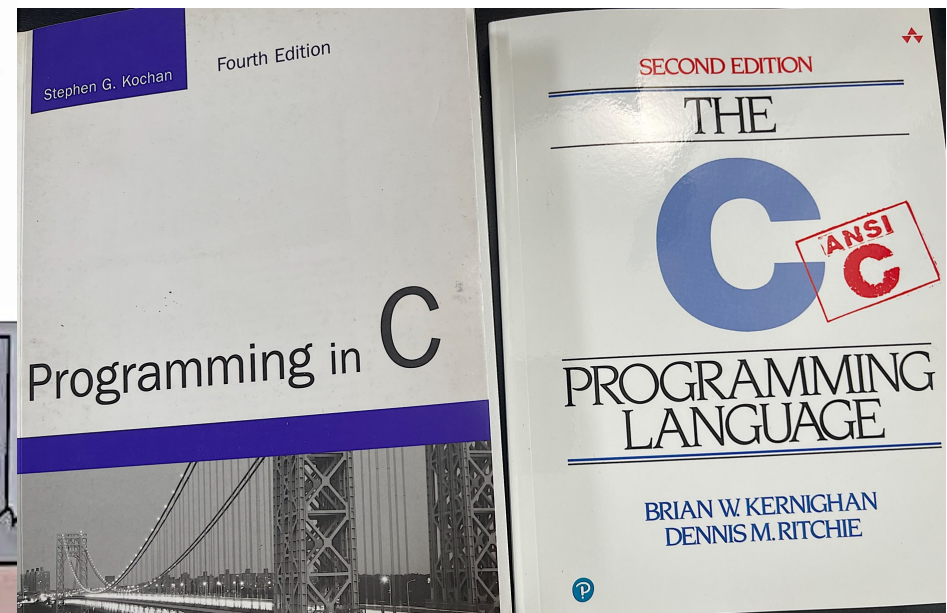


Lecture 15: C part VIII



junior C developer

Suggested Literature



senior C developer

Working with files in C

C has specific libraries to open files directly (instead of relying on bash input / output redirection)

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

// FILE *fopen(const char *file_name,
//             const char *mode_of_operation);
// -----
// modes of operation:
// - read only ("r")
// - write only ("w").
//   Original contents are overwritten
// - appending ("a")
// - read and write ("r+")
// - write but also read the content created
//   by writting ("w+"). Original contents
//   are overwritten
// - reading + appending ("a+")
// - read in binary mode ("rb")
// - writ in binary mode ("wb")

int main()
{
    // Creates a file with access as write-plus mode
    FILE* demo = fopen("demo_file.txt", "w+");

    fprintf(demo, "%s v%0.2f", "Welcome to PHY504", 2.0);

    fclose(demo); // closes the file

    return 0;
}
```

```
whovian@phy504:~/host$ gcc file.c -o file
whovian@phy504:~/host$ ./file
whovian@phy504:~/host$ more demo_file.txt
Welcome to PHY504 v2.00
```

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

int main()
{
    char str[20];
    strcpy(str, "Hello, Phy504!\n");

    FILE* fp = fopen("file4test.txt", "w");

    fputs(str, fp);

    fputs(str, stdout); // terminal

    fclose(fp);

    return 0;
}
```

Working with files in C

fputs is an alternative to fprintf

[fputs vs fprintf](#)

```
whovian@phy504:~/host$ gcc file4.c -o file4
whovian@phy504:~/host$ ./file4
Hello, Phy504!
whovian@phy504:~/host$ more file4test.txt
Hello, Phy504!
```

Working with files in C

```
#include <stdio.h>

int main(void)
{
    FILE* ptr = fopen("file5test.txt", "r");

    char names[50];
    char ages[50];
    char cities[50];

    // return value fscanf: number of fields that
    // it successfully converted and assigned
    while (fscanf(ptr, "%s %s %s", names, ages, cities) == 3)
    {
        printf("%s %s %s \n", names, ages, cities);
    }

    return 0;
}
```

```
whovian@phy504:~/host$ gcc file5.c -o file5
whovian@phy504:~/host$ more file5test.txt
NAME      AGE      BORNCITY
Vivian    37      Rio-de-Janeiro
Beyonce   41      Houston
Rihanna   35      Saint-Michael
Nicki-Minaj 40      Saint-James
whovian@phy504:~/host$ ./file5
NAME AGE BORNCITY
Vivian 37 Rio-de-Janeiro
Beyonce 41 Houston
Rihanna 35 Saint-Michael
Nicki-Minaj 40 Saint-James
```

Like **scanf**, **fscanf** returns **int** = number of variables it read successfully. Use with **while** to do format reading of tables

Working with files

```
#include <stdio.h>

// FILE* pointer to stream - can always be
// either the terminal or an actual file
// char *fgets(char *str, int n, FILE* stream)

int main(void)
{
    const int sz = 200;
    char buf[sz];

    // fgets always required a buffer
    // Always be careful with buffer overflow
    // (input string too large)
    // stdin = standard input (terminal)
    fgets(buf, sz, stdin);

    printf("string is: %s\n", buf);

    return 0;
}
```

fgets can be used
with terminal or file

Alternative
char* gets (char* str)
assumes the
terminal output

```
whovian@phy504:~/host$ gcc file3.c -o file3
whovian@phy504:~/host$ ./file3
Hello Phy504
string is: Hello Phy504
```

Working with files

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

int main ()
{
    FILE* ptr = fopen ("unexist.ent", "r");

    if (ptr == NULL)
    {
        printf ("Error: %s \n", strerror(errno));
    }

    return 0;
}
```

How to convert
int errors to
human readable
messages
(in the std
library)?

errno, strerror

```
whovian@phy504:~/host$ gcc file2.c -o file2
whovian@phy504:~/host$ ./file2
Error: No such file or directory
```

Working with files

errno value	Error
1	/* Operation not permitted */
2	/* No such file or directory */
3	/* No such process */
4	/* Interrupted system call */
5	/* I/O error */
6	/* No such device or address */
7	/* Argument list too long */
8	/* Exec format error */
9	/* Bad file number */
10	/* No child processes */
11	/* Try again */
12	/* Out of memory */
13	/* Permission denied */

This way of
handling error is
not safe when
doing
parallelism

Why?
global variable

source: <https://www.geeksforgeeks.org/error-handling-c-programs/>

GNU nano 4.8

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

```
// source: https://www.educba.com/fseek-in-c/
```

```
// int fseek(FILE *stream,  
//          long int offset,  
//          int pos)
```

```
// SEEK_CUR: current position of the ptr of the file.
```

```
// SEEK_END: moves the file ptr to the end of the file.
```

```
// SEEK_SET: moves the file ptr to the beginning of the file
```

```
int main()
```

```
{
```

```
    FILE* fx = fopen("file6test.txt", "r");
```

```
    fseek(fx, 0, SEEK_END);
```

```
    printf("Position of file pointer is : %ld \n", ftell(fx));
```

```
    fseek(fx, 10, SEEK_SET);
```

```
    int ch;
```

```
    printf("Resulting bytes after the 10 chars: ");
```

```
    while((ch = fgetc(fx)) != EOF) // fgetc get chars from file
```

```
    {
```

```
        putchar(ch);
```

```
    }
```

```
    return 0;
```

```
}
```

```
whovian@phy504:~/host$ more file6test.txt
```

```
Welcome to Phy504, everyone. Are you learning the C language?
```

```
whovian@phy504:~/host$ gcc file6.c -o file6
```

```
whovian@phy504:~/host$ ./file6
```

```
Position of file pointer is : 62
```

```
Resulting bytes after the 10 chars:  Phy504, everyone. Are you learning the C language?
```

Working with files

fseek allows the user to position the file pointer in different places

```

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>

// source:
// www.fundza.com/c4serious/fileIO_reading_all/4_IO_readall.c

int main(void)
{
    FILE* infile = fopen("file7test.txt", "r");

    fseek(infile, 0L, SEEK_END); // 0L = 0 in long int
    long int numbytes = ftell(infile);
    fseek(infile, 0L, SEEK_SET); // reset the file pos

    // grab sufficient memory for the buffer
    char* buffer = (char*)calloc(numbytes, sizeof(char));

    // copy all the text into the buffer Char sz = 1 byte
    fread(buffer, sizeof(char), numbytes, infile);
    fclose(infile);

    // confirm we have read the file by outputing it
    printf("The file contains this text\n\n%s", buffer);

    free(buffer);
    return 0;
}

```

Working with files

Reading entire file into memory at once (faster)

ftell returns long int (current position of the stream in bytes)

```

whovian@phy504:~/host$ more file7test.txt
This is the test file
Welcome to Phy504!
Here we will load the entire file into memory
whovian@phy504:~/host$ gcc file7.c -o file7
whovian@phy504:~/host$ ./file7
The file contains this text

```

```

This is the test file
Welcome to Phy504!
Here we will load the entire file into memory

```

```

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>

// source:
// www.fundza.com/c4serious/fileIO_reading_all/4_IO_readall.c

int main(void)
{
    FILE* infile = fopen("file7test.txt", "r");

    fseek(infile, 0L, SEEK_END); // 0L = 0 in long int
    long int numbytes = ftell(infile);
    fseek(infile, 0L, SEEK_SET); // reset the file pos

    // grab sufficient memory for the buffer
    char* buffer = (char*)calloc(numbytes, sizeof(char));

    // copy all the text into the buffer
    fread(buffer, sizeof(char), numbytes, infile);
    fclose(infile);

    // confirm we have read the file by outputing it
    printf("The file contains this text\n\n%s", buffer);

    free(buffer);
    return 0;
}

```

Working with files

Reading entire file into memory at once (faster)

fread, fwrite are lower-level functions

```

whovian@phy504:~/host$ more file7test.txt
This is the test file
Welcome to Phy504!
Here we will load the entire file into memory
whovian@phy504:~/host$ gcc file7.c -o file7
whovian@phy504:~/host$ ./file7
The file contains this text

```

```

This is the test file
Welcome to Phy504!
Here we will load the entire file into memory

```

Working with files

fwrite is lower-level functions (deal with bytes).

Can be useful when writing **structs** as well

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

// source:
// https://www.geeksforgeeks.org/readwrite-structure-file-c/

struct person
{
    int id;
    char fname[20];
    char lname[20];
};

int main ()
{
    FILE* outfile = fopen ("file8test.txt", "w");

    struct person input1 = {1, "vivian", "Miranda"};
    struct person input2 = {2, "Wayne", "Hu"};

    // write struct to file
    // to visualize in bash: xxd file8test.txt
    fwrite (&input1, sizeof(struct person), 1, outfile);
    fwrite (&input2, sizeof(struct person), 1, outfile);

    fclose (outfile);
    return 0;
}
```

3 arg = total size
2 arg = size of each elem

```
whovian@phy504:~/host$ gcc file8.c -o file8
whovian@phy504:~/host$ ./file8
whovian@phy504:~/host$ xxd file8test.txt
00000000: 0100 0000 7669 7669 616e 0000 0000 0000  ....vivian.....
00000010: 0000 0000 0000 0000 4d69 7261 6e64 6100  ....Miranda.
00000020: 0000 0000 0000 0000 0000 0000 0200 0000  .....
00000030: 5761 796e 6500 0000 0000 0000 0000 0000  Wayne.....
00000040: 0000 0000 4875 0000 0000 0000 0000 0000  ....Hu.....
00000050: 0000 0000 0000 0000                                     .....
```

```

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

// source:
// https://www.geeksforgeeks.org/readwrite-structure-file-c/

struct person
{
    int id;
    char fname[20];
    char lname[20];
};

int main ()
{
    // read mode
    FILE* infile = fopen("file8test.txt", "r");

    struct person input;
    while(fread(&input, sizeof(struct person), 1, infile))
    {
        printf ("id = %d name = %s %s\n", input.id,
            input.fname, input.lname);
    }

    return 0;
}

```

2 arg = size of each elem
3 arg = total size

return = num elem read

Working with files

fwrite is lower-level functions (deal with bytes).

Can be useful when writing **structs** as well

```

whovian@phy504:~/host$ gcc file9.c -o file9
whovian@phy504:~/host$ ./file9
file9      file9.c
whovian@phy504:~/host$ ./file9
id = 1 name = vivian Miranda
id = 2 name = Wayne Hu

```

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

```
int main()
{
```

```
    const double number = 4.0;
```

```
    const double squareRoot = sqrt(number);
```

```
    const double squared = pow(number, 2);
```

```
    const double cube = pow(number, 3);
```

```
    const double sinnum = sin(number); // number in rads
```

```
    const double cosnum = cos(number); // number in rads
```

```
    const double tannum = tan(number); // number in rads
```

```
    const double lognum = log(number); // natural log
```

```
    const double log10num = log10(number); //log_10
```

```
    printf("Sqrt of %.2lf = %.2lf \n", number, squareRoot);
```

```
    printf("Square of %.2lf = %.2lf \n", number, squared);
```

```
    printf("Cube of %.2lf = %.2lf \n", number, cube);
```

```
    printf("sin(%.2lf) = %.2lf \n", number, sinnum);
```

```
    printf("cos(%.2lf) = %.2lf \n", number, cosnum);
```

```
    printf("tan(%.2lf) = %.2lf \n", number, tannum);
```

```
    printf("log(%.2lf) = %.2lf \n", number, lognum);
```

```
    printf("log10(%.2lf) = %.2lf \n", number, log10num);
```

```
    return 0;
```

```
}
```

Standard Library:

math.h

Many useful math functions

Need manual linking with **-lm**

```
whovian@phy504:~/host$ gcc math.c -o math -lm
whovian@phy504:~/host$ ./math
Sqrt of 4.00 = 2.00
Square of 4.00 = 16.00
Cube of 4.00 = 64.00
sin(4.00) = -0.76
cos(4.00) = -0.65
tan(4.00) = 1.16
log(4.00) = 1.39
log10(4.00) = 0.60
```



```

#include <stdio.h>
#include <float.h>

int main (void)
{
    printf("The max value of float = %.5e\n", FLT_MAX);
    printf("The min value of float = %.5e\n\n", FLT_MIN);
    printf("The max value of double = %.5e\n", DBL_MAX);
    printf("The min value of double = %.5e\n\n", DBL_MIN);
    printf("The min eps of a float num = %.5e\n", FLT_EPSILON);
    printf("The min eps of a double num = %.5e\n\n", DBL_EPSILON);

    // (a == b is a bad idea) even the smallest rounding error
    // will cause two floating point numbers to not be equal,
    // operator== is at high risk for returning false

    const double x = 1.0;
    const double y = 1.0 + DBL_EPSILON/2.0;
    // don't try a == b
    if ((x/y - 1.0) < DBL_EPSILON) ←
    {
        printf("x and y are equal in practice\n");
    }
    else
    {
        printf("x and y are NOT equal\n");
    }
    return 0;
}

```

Standard Library: **float.h**

Limits on **float** / **double** types & proper float number comparison

```

whovian@phy504:~/host$ gcc float1.c -o float1
whovian@phy504:~/host$ ./float1

```

The max value of float = 3.40282e+38

The min value of float = 1.17549e-38

The max value of double = 1.79769e+308

The min value of double = 2.22507e-308

The min eps of a float num = 1.19209e-07

The min eps of a double num = 2.22045e-16

x and y are equal in practice

Standard Library: **limits.h**

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

int main()
{
    printf("The min value of INT = %d\n", INT_MIN);
    printf("The max value of INT = %d\n", INT_MAX);

    printf("The min value of CHAR = %d\n", CHAR_MIN);
    printf("The max value of CHAR = %d\n", CHAR_MAX);

    printf("The min value of LONG = %ld\n", LONG_MIN);
    printf("The max value of LONG = %ld\n", LONG_MAX);

    return 0;
}
```

Important
limits on other
types such as
char, **int** and
long int

```
whovian@phy504:~/host$ nano limits1.c
whovian@phy504:~/host$ gcc limits1.c -o limits1
whovian@phy504:~/host$ ./limits1
The min value of INT = -2147483648
The max value of INT = 2147483647
The min value of CHAR = -128
The max value of CHAR = 127
The min value of LONG = -9223372036854775808
The max value of LONG = 9223372036854775807
```

 Not that
large

Standard Library:

ctype.h

```
#include <stdio.h>
#include <ctype.h>
// source: tutorialspoint.com/

int main (void)
{
    const int var1 = 'd';
    const int var2 = '2';
    const int var3 = '\t';
    const int var4 = ' ';

    if(isalnum(var1)) {
        printf("var1 = %c is alphanumeric\n", var1 );
    }
    else {
        printf("var1 = %c is not alphanumeric\n", var1 );
    }
    if(isalnum(var2)) {
        printf("var2 = %c is alphanumeric\n", var2 );
    }
    else {
        printf("var2 = %c is not alphanumeric\n", var2 );
    }
    if(isalnum(var3)) {
        printf("var3 = %c is alphanumeric\n", var3 );
    }
    else {
        printf("var3 = %c is not alphanumeric\n", var3 );
    }
    if(isalnum(var4)) {
        printf("var4 = %c is alphanumeric\n", var4 );
    }
    else {
        printf("var4 = %c is not alphanumeric\n", var4 );
    }
    return 0;
}
```

```
whovian@phy504:~/host$ nano ctype1.c
whovian@phy504:~/host$ gcc ctype1.c -o ctype1
whovian@phy504:~/host$ ./ctype1
var1 = d is alphanumeric
var2 = 2 is alphanumeric
var3 =      is not alphanumeric
var4 =    is not alphanumeric
```

Check if a type is
alphanumeric or
alphabetic or **numeric** or
printable (additional
options and checks
available on **ctype.h**)

Source: <https://www.tutorialspoint.com>

Standard Library:

ctype.h

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

int main ()
{
    const int var1 = 'd';
    const int var2 = '2';
    const int var3 = '\t';
    const int var4 = ' ';

    if( isalpha(var1) ) {
        printf("var1 = |%c| is an alphabet\n", var1 );
    } else {
        printf("var1 = |%c| is not an alphabet\n", var1 );
    }
    if( isalpha(var2) ) {
        printf("var2 = |%c| is an alphabet\n", var2 );
    } else {
        printf("var2 = |%c| is not an alphabet\n", var2 );
    }
    if( isalpha(var3) ) {
        printf("var3 = |%c| is an alphabet\n", var3 );
    } else {
        printf("var3 = |%c| is not an alphabet\n", var3 );
    }
    if( isalpha(var4) ) {
        printf("var4 = |%c| is an alphabet\n", var4 );
    } else {
        printf("var4 = |%c| is not an alphabet\n", var4 );
    }
    return 0;
}
```

```
whovian@phy504:~/host$ gcc ctype2.c -o ctype2
whovian@phy504:~/host$ ./ctype2
var1 = |d| is an alphabet
var2 = |2| is not an alphabet
var3 = |      | is not an alphabet
var4 = | | is not an alphabet
```

Check if a type is
alphanumeric or
alphabetic or **numeric**
or **printable** (additional
options and checks
available on **ctype.h**)

Source:

https://www.tutorialspoint.com/c_standard_library/c_function_isalpha.htm

Standard Library:

ctype.h

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

int main () {
    const int var1 = 'k';
    const int var2 = '8';
    const int var3 = '\t';
    const int var4 = ' ';

    if( isprint(var1) ) {
        printf("var1 = |%c| can be printed\n", var1 );
    } else {
        printf("var1 = |%c| can't be printed\n", var1 );
    }
    if( isprint(var2) ) {
        printf("var2 = |%c| can be printed\n", var2 );
    } else {
        printf("var2 = |%c| can't be printed\n", var2 );
    }
    if( isprint(var3) ) {
        printf("var3 = |%c| can be printed\n", var3 );
    } else {
        printf("var3 = |%c| can't be printed\n", var3 );
    }
    if( isprint(var4) ) {
        printf("var4 = |%c| can be printed\n", var4 );
    } else {
        printf("var4 = |%c| can't be printed\n", var4 );
    }
    return 0;
}
```

```
whovian@phy504:~/host$ gcc ctype3.c -o ctype3
whovian@phy504:~/host$ ./ctype3
var1 = |k| can be printed
var2 = |8| can be printed
var3 = |      | can't be printed
var4 = | | can be printed
```

Check if a type is
alphanumeric or
alphabetic or **numeric**
or **printable** (additional
options and checks
available on **ctype.h**)

Source:

https://www.tutorialspoint.com/c_standard_library/c_function_isprint.htm

```

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

// source: geeksforgeeks. C program for finding the
// largest int among 3 ints using command line args

int main(int argc, char** argv)
{
    if (argc < 4 || argc > 5)
    { // the program itself is an argument
        printf("Please enter 3 integers only\n");
        printf("You entered %i integers\n", argc-1);
        return 1;
    }
    const int a = atoi(argv[1]);
    const int b = atoi(argv[2]);
    const int c = atoi(argv[3]);

    if (a < 0 || b < 0 || c < 0)
    {
        printf("Please enter only positive values\n");
        return 1;
    }
    if (!(a != b && b != c && a != c))
    {
        printf("Please enter 3 different values\n");
        return 1;
    }
    else
    {
        if (a > b && a > c)
            printf("%d is largest\n", a);
        else if (b > c && b > a)
            printf("%d is largest\n", b);
        else if (c > a && c > b)
            printf("%d is largest\n", c);
    }
    return 0;
}

```



Arguments in the main function

- **int argc**: the number of arguments. The program itself count as an argument

```

whovian@phy504:~/host$ ./commandline1
Please enter 3 integers only
You entered 0 integers
whovian@phy504:~/host$ ./commandline1 1
Please enter 3 integers only
You entered 1 integers
whovian@phy504:~/host$ ./commandline1 1 2
Please enter 3 integers only
You entered 2 integers

```

Source: <https://www.geeksforgeeks.org/find-largest-among-three-different-positive-numbers-using-command-line-argument/>

Arguments in the main function

- **char** argv**: an array of strings. In this case we need to convert them to integers

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

// source: geeksforgeeks. C program for finding the
// largest int among 3 ints using command line args

int main(int argc, char** argv)
{
    if (argc < 4 || argc > 5)
    { // the program itself is an argument
        printf("Please enter 3 integers only\n");
        printf("You entered %i integers\n", argc-1);
        return 1;
    }
    const int a = atoi(argv[1]);
    const int b = atoi(argv[2]);
    const int c = atoi(argv[3]);

    if (a < 0 || b < 0 || c < 0)
    {
        printf("Please enter only positive values\n");
        return 1;
    }
    if (!(a != b && b != c && a != c))
    {
        printf("Please enter 3 different values\n");
        return 1;
    }
    else
    {
        if (a > b && a > c)
            printf("%d is largest\n", a);
        else if (b > c && b > a)
            printf("%d is largest\n", b);
        else if (c > a && c > b)
            printf("%d is largest\n", c);
    }
    return 0;
}
```



```
whovian@phy504:~/host$ ./commandline1 1 2 3
3 is largest
whovian@phy504:~/host$ ./commandline1 1 3 3
Please enter 3 different values
whovian@phy504:~/host$ ./commandline1 1 3 -10
Please enter only positive values
whovian@phy504:~/host$ ./commandline1 50 3 2
50 is largest
```

Source: <https://www.geeksforgeeks.org/find-largest-among-three-different-positive-numbers-using-command-line-argument/>

```

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

// source:
// www.geeksforgeeks.org/time-h-header-file-in-c-with-examples/

int frequency_of_primes(int n)
{ // the number of primes less than the given param
  int freq = n - 1;
  for (int i = 2; i <= n; ++i)
  {
    for (int j = sqrt(i); j > 1; --j)
    {
      if (i % j == 0)
      {
        --freq;
        break;
      }
    }
  }
  return freq;
}

int main()
{
  clock_t t = clock();
  const int f = frequency_of_primes(99999);

  printf("The number of primes lower"
        " than 10,0000 is: %d\n", f);

  t = clock() - t;

  printf("No. of clicks %ld clicks (%f seconds).\n",
        t, ((float) t) / CLOCKS_PER_SEC);

  return 0;
}

```

```

whovian@phy504:~/host$ gcc time1.c -o time1 -lm
whovian@phy504:~/host$ ./time1
The number of primes lower than 10,0000 is: 9592
No. of clicks 92552 clicks (0.092552 seconds).

```

Standard Library:

time.h

Functions to display
local (or GMT) time and
to measure execution
elapsed time

[More examples](#)
[\(including display of](#)
[local / UTC time\) via](#)
[geeksforgeeks](#)

```

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

int main () {
    char str[20];

    strcpy(str, "98993489");
    const int val = atoi(str);
    printf("String value = %s, Int val = %d\n", str, val);

    strcpy(str, "tutorialspoint.com");
    const int val2 = atoi(str); // when failing, returns 0
    printf("String value = %s, Int val = %d\n", str, val2);

    strcpy(str, "2.3");
    const double val3 = atof(str);
    printf("String value = %s, Float val = %lf\n", str, val3);

    strcpy(str, "tutorialspoint.com");
    const double val4 = atof(str); // when failing, returns 0
    printf("String value = %s, Float val = %lf\n", str, val4);

    strcpy(str, "9899348912434");
    const long int val5 = atol(str);
    printf("String value = %s, Long int val = %ld\n", str, val5);

    strcpy(str, "tutorialspoint.com");
    const long int val6 = atol(str); // when failing, returns 0
    printf("String value = %s, Float val = %ld\n", str, val6);

    return 0;
}

```

```

whovian@phy504:~/host$ ./stdlib1
String value = 98993489, Int val = 98993489
String value = tutorialspoint.com, Int val = 0
String value = 2.3, Float val = 2.300000
String value = tutorialspoint.com, Float val = 0.000000
String value = 9899348912434, Long int val = 9899348912434
String value = tutorialspoint.com, Float val = 0

```

Standard Library:

stdlib.h

Many important functions

- **malloc/calloc/realloc**
- **free**
- **exit(), abort()**
- **int rand()**
- **atof, atoi, atoll**
- **char* getenv(const char *name)**

Source:

https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm

```

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

// double strtod(const char *str, char **endptr)
// str: string to be converted to a value
// endptr: reference to an already allocated object of type,
//         char*, whose value is set by the function to the
//         next char in str after the numerical value.

int main ()
{
    char str[30] = "20.30300 This is test";
    char *ptr;

    double num1 = strtod(str, &ptr);

    printf("The number(double) is %lf\n", num1);
    printf("String part is |%s|\n", ptr);

    double num2 = atof(str);
    printf("The number(double) via atof is %lf\n", num2);

    return(0);
}

```

```

whovian@phy504:~/host/C$ ./stdlib5
The number(double) is 20.303000
String part is | This is test|
The number(double) via atof is 20.303000

```

Standard Library: **stdlib.h**

Functions **atof**, **atoi**,
atoll work ok but
there are functions
(e.g., **strtod**) that are
more complete.

strtod provides mixed
translation from string
to number and words

Source:

https://www.tutorialspoint.com/c_standard_library/c_function_strtod.htm

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

```
// double strtod(const char *str, char **endptr)
// this function (same for strtod) allows error
// checking (that is not the case of atod/atoi)
```

```
int main ()
```

```
{
```

```
    char str[30] = "This is test";
```

```
    char *ptr;
```

```
    double num1 = strtod(str, &ptr);
```

```
    if (ptr == str) // how to check errors
```

```
    {
```

```
        printf("Conversion of |%s| to double fail\n", str);
```

```
    }
```

```
    char str2[30] = "1234.0 This is test";
```

```
    char *ptr2;
```

```
    double num2 = strtod(str2, &ptr2);
```

```
    if (ptr2 != str2) // how to check errors
```

```
    {
```

```
        printf("Conversion of |%s| to double %lf succeed\n",
               str, num2);
```

```
    }
```

```
    return 0;
```

```
}
```

Standard

Library: **stdlib.h**

strtod/strtol

functions also
provide a way to
check if the
conversion was
successful

```
whovian@phy504:~/host/C$ gcc stdlib7.c -o stdlib7
```

```
whovian@phy504:~/host/C$ ./stdlib7
```

```
Conversion of |This is test| to double fail
```

```
Conversion of |This is test| to double 1234.000000 succeed
```

```

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

// long int strtol(const char *str, char **endptr, int base)

// str: string to be converted to a value

// endptr: reference to an already allocated object of type,
//          char*, whose value is set by the function to the
//          next char in str after the numerical value.

// base:    This is the base, which must be between 2 and 36
//          inclusive, or be the special value 0.

int main ()
{
    char str[30] = "-2030300 This is test";
    char *ptr;

    long int num1 = strtol(str, &ptr, 10);

    printf("The number (long int) is %ld\n", num1);
    printf("String part is |%s|\n", ptr);

    long int num2 = atof(str);
    printf("The number (long int) via atoi is %ld\n", num2);

    return 0;
}

```

```

whovian@phy504:~/host/C$ ./stdlib6
The number (long int) is -2030300
String part is | This is test|
The number (long int) via atoi is -2030300

```

Standard Library:

stdlib.h

Functions **atof**, **atoi**, **atoll** work ok but there are functions (e.g., **strtol**) that are more complete

strtod provides mixed translation from string to number and words

Source:

https://www.tutorialspoint.com/c_standard_library/c_function_strtol.htm

Standard Library: **stdlib.h**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
// source:
// https://www.tutorialspoint.com/
// c_standard_library/c_function_rand.htm

int main (void)
{
    time_t t;
    // Intializes random number generator
    srand((unsigned) time(&t));

    for(int i=0; i<5 ; i++ )
    { // Print 5 random number [0,49]
        printf("%d\n", rand() % 50);
    }

    return 0;
}
```

stdlib.h provides a high-quality random number generator

- **srand** = provide the seed for the generator

```
whovian@phy504:~/host$ gcc stdlib2.c -o stdlib2
whovian@phy504:~/host$ ./stdlib2
27
26
27
19
44
whovian@phy504:~/host$ ./stdlib2
46
8
29
48
30
```

Source: https://www.tutorialspoint.com/c_standard_library/stdlib_h.htm

Standard Library: **stdlib.h**

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

int main ()
{
    printf("PATH : %s\n", getenv("PATH"));
    printf("HOME : %s\n", getenv("HOME"));
    printf("ROOT : %s\n", getenv("ROOT"));

    return 0;
}
```

stdlib.h provides the **getenv** function that allows **shell/bash** environmental variables to be read

```
whovian@phy504:~/host/C$ gcc stdlib3.c -o stdlib3
whovian@phy504:~/host/C$ ./stdlib3
PATH : /opt/conda/bin:/usr/local/nvidia/bin:/usr/local/cuda/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
HOME : /home/whovian
ROOT : (null)
```

Source: https://www.tutorialspoint.com/c_standard_library/c_function_getenv.htm

```

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

// void qsort(void *base, size_t nitems, size_t size,
// int (*compar)(const void *, const void*))

int values[] = { 88, 56, 100, 2, 25 };

int cmpfunc (const void* a, const void* b)
{
    return ( *(int*)a - *(int*)b );
}

int main ()
{
    printf("Before sorting the list is:\n");
    for(int n=0; n<5; n++)
    {
        printf("%d ", values[n]);
    }
    printf("\n");

    qsort(values, 5, sizeof(int), cmpfunc);

    printf("After sorting the list is:\n");
    for (int n=0; n<5; n++)
    {
        printf("%d ", values[n]);
    }
    printf("\n");
    return 0;
}

```

Standard Library:

stdlib.h

stdlib.h provide generic functions such as **qsort**.

Generic functions use void pointers because anything can be casted into **void***

```

whovian@phy504:~/host/C$ gcc stdlib4.c -o stdlib4
whovian@phy504:~/host/C$ ./stdlib4
Before sorting the list is:
88 56 100 2 25
After sorting the list is:
2 25 56 88 100

```

Source:

https://www.tutorialspoint.com/c_standard_library/c_function_qsort.htm

Headers in C

```
#include <stdio.h>

void print_int(const int x)
{
    printf("The number is: %i\n", x);
}
```

```
#ifndef __PHY504_TEST_HEADER_H
#define __PHY504_TEST_HEADER_H
#ifdef __cplusplus
extern "C" {
#endif

void print_int(const int x);

// declaration does not care about
// variable names (only care about type).
// So you can declare the function as
// void print_int(const int);

#ifdef __cplusplus
}
#endif
#endif // HEADER GUARD
```

File 1: **testheader1.c**

body of the function

header 1: **testheader1.h**

Include declarations of
functions (and global
variables / structs)

Headers need pre-
processor guards

Headers in C

```
#include <stdio.h>

void print_int(const int x)
{
    printf("The number is: %i\n", x);
}
```

```
#ifndef __PHY504_TEST_HEADER_H
#define __PHY504_TEST_HEADER_H
#ifdef __cplusplus
extern "C" {
#endif

void print_int(const int x);

// declaration does not care about
// variable names (only care about type).
// So you can declare the function as
// void print_int(const int);

#ifdef __cplusplus
}
#endif
#endif // HEADER GUARD
```

File 1: **testheader1.c**

body of the function

Advanced technical point

If you mix C and C++ code, the C headers need these additional guards.

Why? Differences between C and C++ on something called “Name Mangling”

Headers in C

```
#include "testheader1.h"
```

```
int main(void)
{
    const int myint = 10;
    print_int(myint);
    return 0;
}
```

File 2: **testheader1_main.c**

Location of the main
function

Why the double quotes in “**testheader1.h**”?

Trick that tells the pre-processor to look for the file in the
current directory first

How do I compile the code? No need to include the header

```
whovian@phy504:~/host$ gcc testheader1_main.c testheader1.c -o testheader
whovian@phy504:~/host$ ./testheader
The number is: 10
```

Static variables outside functions

```
#include <stdio.h>

#ifndef __PHY504_TEST_HEADER3_H
#define __PHY504_TEST_HEADER3_H
#ifdef __cplusplus
extern "C" {
#endif

// When you #include "testheader3.h" in each of a dozen files,
// the compiler will produce a new instance of the max function
// in each of them. But because you've given the function internal
// linkage, none of the files has made public the function name max,
// so all dozen separate instances of the function can live
// independently with no conflicts.

static inline void print_int(const int x)
{
    printf("The number is: %i\n", x);
}

#ifdef __cplusplus
}
#endif
#endif // HEADER GUARD
```

- Internal linkage = file's instance of a var x or a function f() is its own and matches only other instances of x or f() in the same scope. Use the static keyword to indicate internal linkage.

Source:

<https://stackoverflow.com/questions/7762731/whats-the-difference-between-static-and-static-inline-function>

Source2:

<https://stackoverflow.com/a/25000931>

Static variables outside functions

```
27  static int GSL_WORKSPACE_SIZE = 250;
28  static int use_linear_ps_limber = 0; // 0 or 1
29
30  static int include_HOD_GX = 0; // 0 or 1
31
32  static int include_RSD_GS = 0; // 0 or 1
33  static int include_RSD_GG = 1; // 0 or 1
34  static int include_RSD_GK = 0; // 0 or 1
35  static int include_RSD_GY = 0; // 0 or 1
36
```

Example of static global variables with internal linkage used in my research. They are great for setting compile-time options that are relevant only in one file

Source: https://github.com/CosmoLike/cocoa/blob/main/Cocoa/external_modules/code/cosmolike/cosmo2D.c

Static variables outside functions

GNU nano 6.2 teststatic2.c *

```
#include <stdio.h>

static int myglobal = 10;

void printglobal()
{
    printf("myglobal: %i\n", myglobal);
    myglobal += 50;
}
```

```
#include <stdio.h>
#include "teststatic2.h"
#include "teststatic3.h"
```

```
int main ()
{
    printglobal();
    printglobal2();

    printglobal();
    printglobal2();
}
```

GNU nano 6.2 teststatic3.c

```
#include <stdio.h>

static int myglobal = 20;

void printglobal2()
{
    printf("myglobal: %i\n", myglobal);
    myglobal++;
}
```

```
whovian@phy504:~/host/C$ gcc teststatic.c teststatic2.c teststatic3.c -o teststatic
whovian@phy504:~/host/C$ ./teststatic
myglobal: 10
myglobal: 20
myglobal: 60
myglobal: 21
```

They have the same name, but act independent because of the **static** prefix

Static variables outside functions

```
GNU nano 6.2 teststatic2.c *
#include <stdio.h>

int myglobal = 10;

void printglobal()
{
    printf("myglobal: %i\n", myglobal);
    myglobal += 50;
}
```

```
GNU nano 6.2 teststatic3.c *
#include <stdio.h>

int myglobal = 20;

void printglobal2()
{
    printf("myglobal: %i\n", myglobal);
    myglobal++;
}
```

```
#include <stdio.h>
#include "teststatic2.h"
#include "teststatic3.h"
```

```
int main ()
{
    printglobal();
    printglobal2();

    printglobal();
    printglobal2();
}
```

```
whovian@phy504:~/host/C$ gcc teststatic.c teststatic2.c teststatic3.c -o teststatic
/usr/bin/ld: /tmp/ccs3VJzB.o(.data+0x0): multiple definition of `myglobal'; /tmp/ccnekUmq.o(.data+0x0): first defined here
collect2: error: ld returned 1 exit status
```

Without **static** prefix, linking of the files
done by the compile fail

How to make a global variable to all files?

```
GNU nano 6.2 teststatic4.h *
#ifndef PHY504_TESTSTATIC4_H
#define PHY504_TESTSTATIC4_H

void printglobal();

extern int myglobal;

#endif
```

```
GNU nano 6.2 teststatic4.c
#include <stdio.h>

int myglobal = 10;

void printglobal()
{
    printf("myglobal: %i\n", myglobal);
    myglobal += 50;
}
```

```
GNU nano 6.2 teststatic5.h *
#ifndef PHY504_TESTSTATIC5_H
#define PHY504_TESTSTATIC5_H

void printglobal2();

#endif
```

```
GNU nano 6.2 teststatic5.c *
#include <stdio.h>
#include "teststatic4.h"

void printglobal2()
{
    printf("myglobal: %i\n", myglobal);
    myglobal++;
}
```

```
GNU nano 6.2
#include <stdio.h>
#include "teststatic4.h"
#include "teststatic5.h"

int main ()
{
    printglobal();
    printglobal2();

    printglobal();
    printglobal2();
}
```

```
whovian@phy504:~/host/C$ gcc teststatic4.c teststatic5.c teststatic6.c -o teststatic6
whovian@phy504:~/host/C$ ./teststatic6
myglobal: 10
myglobal: 60
myglobal: 61
myglobal: 111
```

Inline functions - Headers in C

```
#include <stdio.h>

#ifndef __PHY504_TEST_HEADER2_H
#define __PHY504_TEST_HEADER2_H
#ifdef __cplusplus
extern "C" {
#endif

extern inline void print_int(const int x)
{
    printf("The number is: %i\n", x);
}

#ifdef __cplusplus
}
#endif
#endif // HEADER GUARD
```

- inline instructs the compiler to **attempt to** embed the function content into the calling code
- For small functions that are called frequently that can make a big performance difference.
- This is only a "hint", and the compiler may ignore it. From header also require **extern** for linkage

Source:

<https://stackoverflow.com/questions/7762731/whats-the-difference-between-static-and-static-inline-function>

Source2: <https://stackoverflow.com/a/25000931>

Inline functions - Headers in C

```
#include <stdio.h>

#ifndef __PHY504_TEST_HEADER2_H
#define __PHY504_TEST_HEADER2_H
#ifdef __cplusplus
extern "C" {
#endif

extern inline void print_int(const int x)
{
    printf("The number is: %i\n", x);
}

#ifdef __cplusplus
}
#endif
#endif // HEADER GUARD
```

This is only a "hint", and the compiler may ignore it. C also requires **extern** for linkage. Why?

External linkage means that symbols that match across files should be treated as the same thing by the linker. The **extern** keyword will be useful to indicate external linkage

Source:

<https://stackoverflow.com/questions/7762731/whats-the-difference-between-static-and-static-inline-function>

Source2: <https://stackoverflow.com/a/25000931>

Static inline functions

```
#include <stdio.h>

#ifndef __PHY504_TEST_HEADER3_H
#define __PHY504_TEST_HEADER3_H
#ifdef __cplusplus
extern "C" {
#endif

// When you #include "testheader3.h" in each of a dozen files,
// the compiler will produce a new instance of the max function
// in each of them. But because you've given the function internal
// linkage, none of the files has made public the function name max,
// so all dozen separate instances of the function can live
// independently with no conflicts.

static inline void print_int(const int x)
{
    printf("The number is: %i\n", x);
}

#ifdef __cplusplus
}
#endif
#endif // HEADER GUARD
```

- Internal linkage = file's instance of a var **x** or a function **f()** is its own and matches only other instances of **x** or **f()** in the same scope (file).

Use the static keyword to indicate internal linkage.

Source:

<https://stackoverflow.com/questions/7762731/whats-the-difference-between-static-and-static-inline-function>

Source2: <https://stackoverflow.com/a/25000931>

Static inline functions

```
GNU nano 6.2 testheader4.h
#include <stdio.h>

#ifndef __PHY504_TEST_HEADER4_H
#define __PHY504_TEST_HEADER4_H

static inline void print_int(const int x)
{
    printf("Print1: The number is: %i\n", x);
}

#endif // HEADER GUARD
```

```
GNU nano 6.2 testheader6.h
#include <stdio.h>

#ifndef __PHY504_TEST_HEADER6_H
#define __PHY504_TEST_HEADER6_H

static inline void print_int(const int x)
{
    printf("Print2: The number is: %i\n", x);
}

#endif // HEADER GUARD
```

```
GNU nano 6.2 testheader5.h
#ifndef __PHY504_TEST_HEADER5_H
#define __PHY504_TEST_HEADER5_H

void test_static_inline1();

#endif // HEADER GUARD
```

```
GNU nano 6.2 testheader6.c
#include "testheader5.h"
#include "testheader6.h"

int main()
{
    print_int(100);
    test_static_inline1();
}
```

```
GNU nano 6.2 testheader5.c
#include "testheader4.h"

void test_static_inline1()
{
    print_int(10);
}
```

```
whovian@phy504:~/host/C$ gcc testheader5.c testheader6.c -o testheader6
whovian@phy504:~/host/C$ ./testheader6
Print2: The number is: 100
Print1: The number is: 10
```

Static inline functions

Without the static keyword, compiler can't link the files

```
whovian@phy504:~/host/C$ gcc testheader8.c testheader9.c -o testheader6
/usr/bin/ld: /tmp/cc0Tki0j.o: in function `test_static_inline1':
testheader8.c:(.text+0xe): undefined reference to `print_int'
/usr/bin/ld: /tmp/ccW2WQF7.o: in function `main':
testheader9.c:(.text+0xe): undefined reference to `print_int'
collect2: error: ld returned 1 exit status
```

How to create a single inline function? Keyword **extern**

```
GNU nano 6.2 testheader10.h
#include <stdio.h>

#ifndef __PHY504_TEST_HEADER10_H
#define __PHY504_TEST_HEADER10_H

extern inline void print_int(const int x)
{
    printf("The number is: %i\n", x);
}

#endif // HEADER GUARD
```

```
GNU nano 6.2 testheader10.c
#include "testheader10.h"

int main()
{
    print_int(100);
}
```

```
whovian@phy504:~/host/C$ gcc testheader10.c -o testheader10
whovian@phy504:~/host/C$ ./testheader10
The number is: 100
```

Casting between const and non-const

```
#include <stdio.h>

// casting between const and not const
// not a good design but possible

// source: 21st Century C, 2nd Edition.
// Author: Ben Klemens
void set_eltmt(int* a, int* b)
{
    a[0] = 3;
}

int main()
{
    int a[10] = {0,1,2,3,4,5,6,7,8,9};
    const int b[10] = {0,1,2,3,4,5,6,7,8,9};
    const int* c = b; // variable ptr to a const int

    set_eltmt(a, (int*) c);

    printf("a[0] = %i, a[1] = %i\n", a[0], a[1]);
    printf("b[0] = %i, b[1] = %i\n", b[0], b[1]);
    return 0;
}
```

When working with functions (especially external functions), the const can cause API incompatibilities.

Casting is possible if the function don't break the const promise (otherwise you get undefined behavior)

```
whovian@phy504:~/host/C$ gcc castingconst.c -o castingconst
whovian@phy504:~/host/C$ ./castingconst
a[0] = 3, a[1] = 1
b[0] = 0, b[1] = 1
```

Casting between const and non-const

```
#include <stdio.h>

// casting between const and not const
// not a good design but possible

// source: 21st Century C, 2nd Edition.
// Author: Ben Klemens
void set_elmt(int* a, int* b)
{
    a[0] = 3;
}

int main()
{
    int a[10] = {0,1,2,3,4,5,6,7,8,9};
    const int b[10] = {0,1,2,3,4,5,6,7,8,9};
    const int* c = b; // variable ptr to a const int

    set_elmt((int*) c, a); // casting from non-const to
                           // const is always ok

    printf("a[0] = %i, a[1] = %i\n", a[0], a[1]);
    printf("b[0] = %i, b[1] = %i\n", b[0], b[1]);
    return 0;
}
```

```
whovian@phy504:~/host/C$ ./castingconst2
a[0] = 0, a[1] = 1
b[0] = 3, b[1] = 1
whovian@phy504:~/host/C$
```

Seems to work fine with our gcc version but it is undefined behavior. Why?

Sometimes const variables sits in unwritable memory location (memory with read-only attributes)

Source:

<https://stackoverflow.com/a/1704922/2472169>



Homework

- Write a program that converts T (input from the terminal – main args) degrees Fahrenheit to Celsius using the formula $C = 5 (F - 32) / 9$
- Write a program to evaluate the polynomial shown here $3x^3 - 5x^2 + 6$ for x (input from the terminal – use main args), and display the results in scientific notation.
- Write a program that computes the numerical integration via the trapezoidal rule of a function (example $f(x) = x^2$ from 0 to 10). Now loop that integral in an outer for loop so the calculation is repeated 1000 times. Use **time.h** to compute the execution time of your program.
- Write a program that converts T (input from terminal – main args) degrees Fahrenheit to Celsius using the formula $C = 5 (F - 32) / 9$. Now save this function in a different C file and write a header so the function can be called from **main.c**, where you will convert of a few Ts in the main function.

Congratulations

You now have a solid understanding of the C language, which can be incredibly helpful in scientific research (C is fast and reliable)

