

# C Programming

---

A crash course for C++ programmers

1

## C vs. C++: Differences

---

- C does not have classes/objects!
  - all code is in functions (subroutines).
- C structures can not have methods
- C I/O is based on library functions:
  - printf, scanf, fopen, fclose, fread, fwrite, ...

2

## C vs. C++: Differences (cont.)

---

- C does not support any function overloading (you can't have 2 functions with the same name).
- C does not have **new** or **delete**, you use **malloc()** and **free()** library functions to handle dynamic memory allocation/deallocation.
- C does not have *reference variables*

3

## C vs. C++: Similarities

---

- Built-in data types: **int**, **double**, **char**, etc.
- Preprocessor (handles **#include**, **#define**, etc.)
- Control structures: **if**, **while**, **do**, **for**, etc.
- Operators: **+** **-** **\*** **/** **==** **!=** **<** **>** **+=** **++** etc.

4

## C vs. C++: Similarities (cont.)

- There must be a function named `main()`.
- function definitions are done the same way.
- Can split code in to files (object modules) and link modules together.

5

## Simple C Program

```
#include <stdio.h>

int main(void) {
    printf("Hello World\n");
    return(0);
}
```

6

## Another Program

```
#include <stdio.h>

void printhello(int n) {
    int i;
    for (i=0;i<n;i++)
        printf("Hello World\n");
}

void main() {
    printhello(5);
}
```

7

## Typical C Program

includes →

defines, data type definitions, global variable declarations →

function definitions →

`main()` →

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 1000
typedef char bigstring[MAX];
char *reverse(char *) {
    char buff[MAX];
    int iLen;
    len = strlen(s);
    printf("reversing %s\n",s);
    for (i=0;i<len;i++)
        buff[i] = s[len-i-1];
    buff[i] = '\0';
    strcpy(buff,s);
    return(s);
}

void main(int argc,char **argv) {
    if (argc<2) {
        printf("Invalid usage - must supply a string\n");
        exit(0);
    }
    printf("%s\n",reverse(argv[1]));
}
```

8

## A Real C Program

---

- Program that accepts one command line argument.
- Treats the command line argument as a string, and reverses the letters in the string.
- Prints out the result (the reversed string).

9

## reverse.c - part 1

---

```
#include <stdio.h> /* printf */
#include <stdlib.h> /* malloc,free */

/* MAX is the size of the largest string
   we can handle */
#define MAX 1000

/* bigstring is a new data type */
typedef char bigstring[MAX];
```

10

## reverse.c - part 2

---

```
/* reverses a string in place
   returns a pointer to the string
*/
char *reverse( char *s) {
    bigstring buf;
    int i,len;
    len = strlen(s); /* find the length */
    for (i=0;i<len;i++)
        buf[i] = s[len-i-1];
    buf[i]='\0'; /* null terminate!*/
    strcpy(s,buf); /* put back in to s */
    return(s);
}
```

11

## reverse.c - part 3

---

```
void main(int argc,char **argv) {
    if (argc<2) {
        printf("Invalid usage - must supply
a string\n");
        exit(0);
    }
    printf("%s\n",reverse(argv[1]));
}
```

12

## Using dynamic allocation

---

```
char *reverse( char *s) {
    char *buf;
    int i,len;
    len = strlen(s);

    /* allocate memory */
    buf = malloc(len+1);

    for (i=0;i<len;i++)
        buf[i] = s[len-i-1];
    buf[i]='\0';
    strcpy(s,buf); free(buf);
    return(s);
}
```

13

## Compiling on Unix

---

Traditionally the name of the C compiler that comes with Unix is "cc".

We can use the Gnu compiler named "gcc".

```
gcc -o reverse reverse.c
```

tells the compiler to create executable file with the name **reverse**

tells the compiler the name of the input file.

14

## Running the program

---

```
>./reverse hidave
evadih

>./reverse This is a long string
sihT

>./reverse "This is a long string"
gnirts gnol a si sihT
```

15

## C Libraries

---

- Standard I/O: printf, scanf, fopen, fread, ...
- String functions: strcpy, strspn, strtok, ...
- Math: sin, cos, sqrt, exp, abs, pow, log,...
- System Calls: fork, exec, signal, kill, ...

16

## Quick I/O Primer - printf

---

```
int printf( const char *, . . . );
```

. . . means “variable number of arguments”.

The first argument is required (a string).

Given a simple string, printf just prints the string (to *standard output*).

17

## Simple printf

---

```
printf("Hi Dave - I am a string\n");
```

```
printf("I\\thave\\ttabs\n");
```

```
char s[100];  
strcpy(s,"printf is fun!\n");  
printf(s);
```

18

## arguing with printf

---

You can tell `printf` to embed some values in the string – these values are determined at run-time.

```
printf("here is an integer: %d\n",i);
```

the `%d` is replaced by the value of the argument following the string (in this case `i`).

19

## More integer arguments

---

```
printf("%d + %d = %d\n",x,y,x+y);
```

```
for (j=99;j>=0;j--)  
    printf("%d bottles of beer on the wall\n");
```

```
printf("%d is my favorite number\n",17);
```

20

## printf is dumb

---

- `%d` is replaced by the value of the parameter when treated as a integer.
- If you give `printf` something that is not an integer – it doesn't know!

```
printf("Print an int %d\n","Hi Dave");
```

```
Print an int 134513884
```

21

## Other formats

---

- `%d` is a format – it means “treat the parameter as an integer”
- `%s` means “treat it as a string”
- `%c` is for characters (**char**)
- `%f` is for floating point numbers
- `%%` means print a single `'%'`

22

## Fun with printf

---

```
char *s = "Hi Dave";  
printf("The string \"%s\" is %d  
characters long\n",  
s,strlen(s));  
  
printf("The square root of 10 is  
%f\n",sqrt(10));
```

23

## Controlling the output

---

- There are formatting options that you can use to control field width, precision, etc.

```
printf("The square root of 10 is  
%20.15f\n",sqrt(10));
```

```
The square root of 10 is  
3.162277660168380
```

24



## scanf

---

```
int scanf(const char *format, ...);
```

- Remember ... means “variable number of arguments” (good thing you have a memory).
- Looks kinda like `printf`

29

## What scanf does

---

- Uses format string to determine what kind of variable(s) it should read.
- The arguments are the *addresses* of the variables.

```
scanf("%d %d",&x,&y);
```

30

## A simple example of scanf

---

```
float x;  
  
printf("Enter a number\n");  
scanf("%f",&x);  
  
printf("The square root of %f is  
%f\n",x,sqrt(x));
```

31

## scanf and strings

---

Using `%s` in a `scanf` string tells `scanf` to read the next *word* from input – NOT a line of input:

```
char s[100];  
printf("Type in your name\n");  
scanf("%s",&s);  
printf("Your name is %s\n",s);
```

32

## man scanf

- Check out the man page for more details.

33

## Reading a line

---

- You can use the function `fgets` to read an entire line:

```
char *fgets(char *s, int size,  
            FILE *stream);
```

`size` is the maximum # chars

`FILE` is a *file handle*

34

## Using `fgets` to read from `stdin`

---

```
char s[101];  
  
printf("Type in your name\n");  
fgets(s,100,stdin);  
  
printf("Your name is %s\n",s);
```

35

## Other I/O stuff

---

- `fopen, fclose`
- `fscanf, fprintf, fgets`
- `fread, fwrite`
- Check the man pages for the details.

36

## String functions

---

```
char *strcpy(char *dest,  
             const char *src);  
  
size_t strlen(const char *s);  
  
char *strtok(char *s,  
             const char *delim);
```

37

## Math library

---

- The math library is usually provided as an external library (not as part of the *standard C library*).
- You must tell the compiler you want the math library:

```
gcc -o myprog myprog.c -lm
```

↑  
means "add in the math library"

38

## System Calls

---

- The C library includes functions that make calls to subroutines that are actually part of the operating system.
- We will look at how to use these functions when we discuss Unix.

39