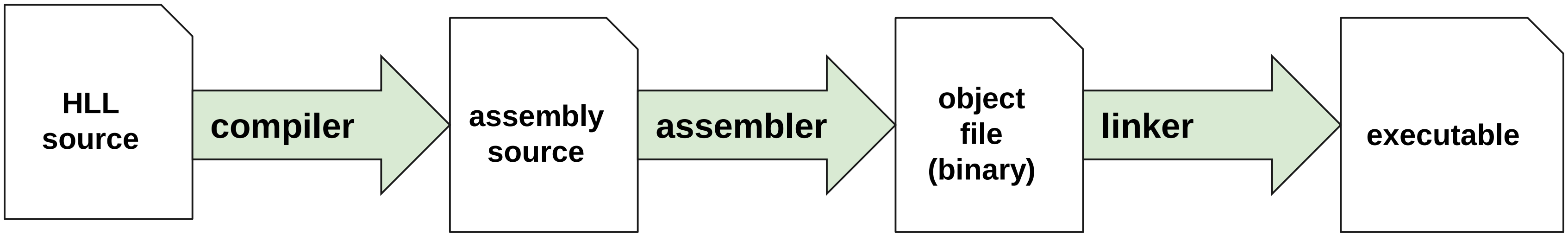
Linux System Programming Part 2 - Programming

IBA Bulgaria
2018

The compilation process



HLL source

compiler

assembly source

assembler

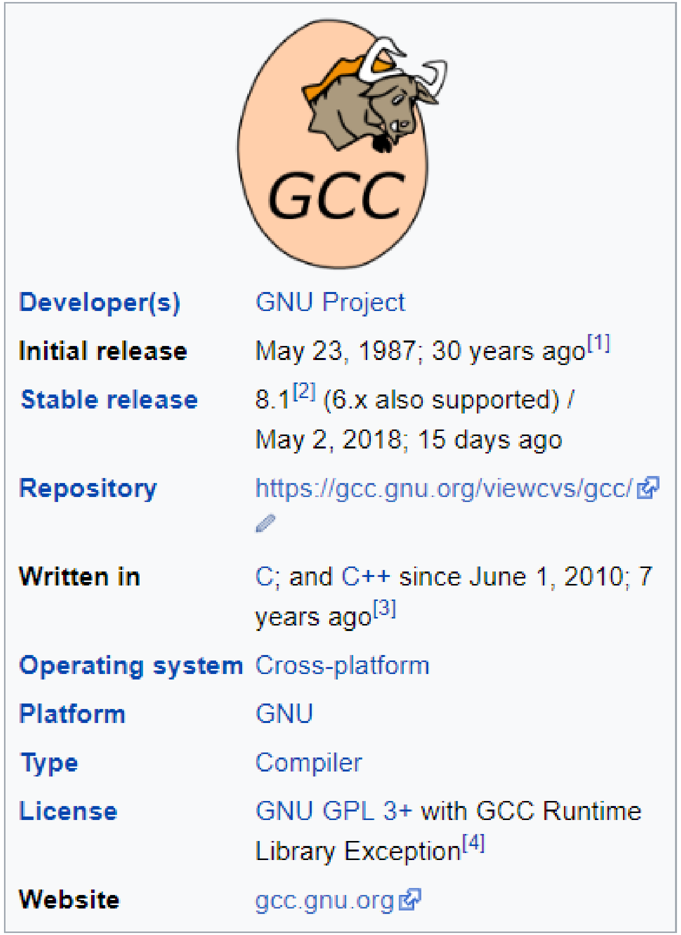
object file (binary)

linker

executable

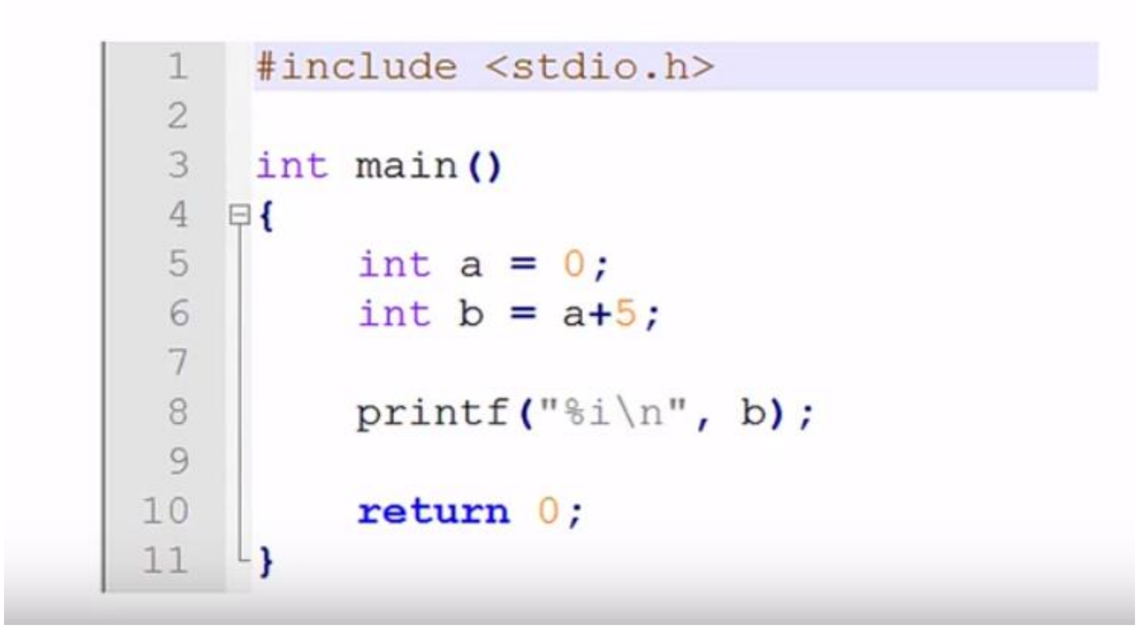
The GNU Compiler Collection (GCC)

- A **compiler system** produced by the GNU Project supporting various programming languages.
- Originally named the **GNU C C**ompiler, when it only handled the **C** programming language.
- We will use it to make our **C** sources into executables.



Developer(s)	GNU Project
Initial release	May 23, 1987; 30 years ago[1]
Stable release	8.1[2] (6.x also supported) / May 2, 2018; 15 days ago
Repository	https://gcc.gnu.org/viewcvs/gcc/
Written in	C; and C++ since June 1, 2010; 7 years ago[3]
Operating system	Cross-platform
Platform	GNU
Type	Compiler
License	GNU GPL 3+ with GCC Runtime Library Exception[4]
Website	gcc.gnu.org

Example code (first.c)

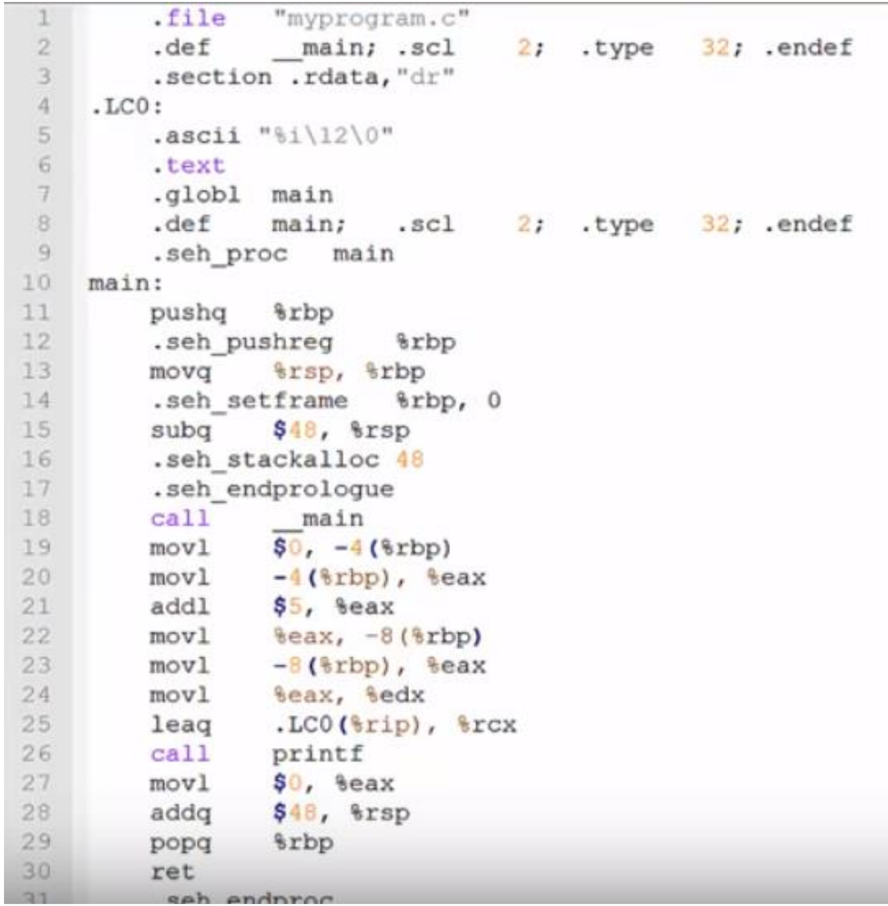


```c
#include <stdio.h>

int main()
{
    int a = 0;
    int b = a+5;

    printf("%i\n", b);

    return 0;
}
```

Compile to assembly

gcc -S first.c

- ‘**-S**’: Stop after the stage of compilation proper; do not assemble.
- The output is ‘**first.s**’.

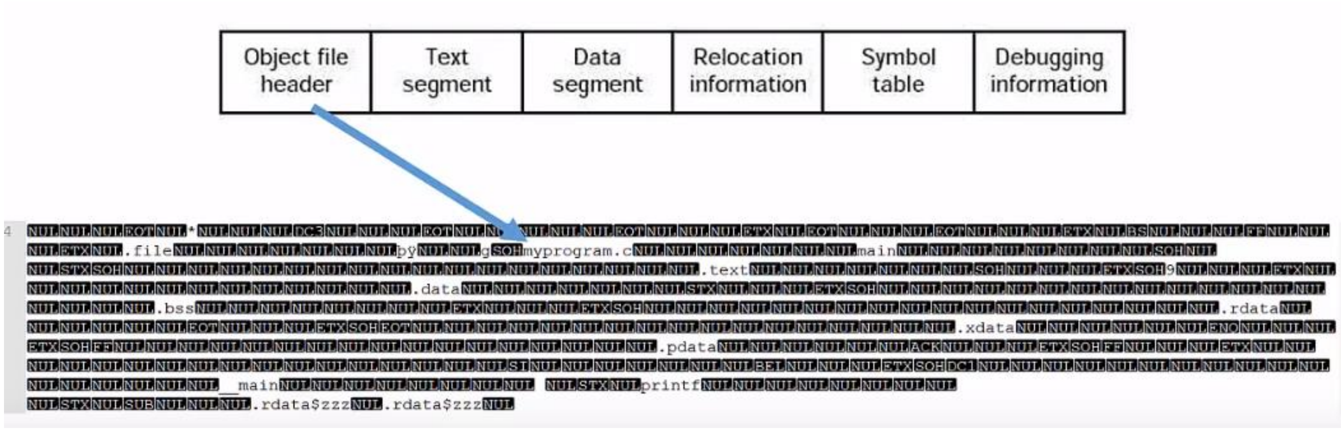


```
	.file	"myprogram.c"
	.def	__main;	.scl	2;	.type	32;	.endef
	.section .rdata,"dr"
.LC0:
	.ascii "%i\12\0"
	.text
	.globl	main
	.def	main;	.scl	2;	.type	32;	.endef
	.seh_proc	main
main:
	pushq	%rbp
	.seh_pushreg	%rbp
	movq	%rsp, %rbp
	.seh_setframe	%rbp, 0
	subq	$48, %rsp
	.seh_stackalloc	48
	.seh_endprologue
	call	__main
	movl	$0, -4(%rbp)
	movl	-4(%rbp), %eax
	addl	$5, %eax
	movl	%eax, -8(%rbp)
	movl	-8(%rbp), %eax
	movl	%eax, %edx
	leaq	.LC0(%rip), %rcx
	call	printf
	movl	$0, %eax
	addq	$48, %rsp
	popq	%rbp
	ret
	.seh_endproc
```

Compile and assemble to object file

gcc -c first.c

- first.o

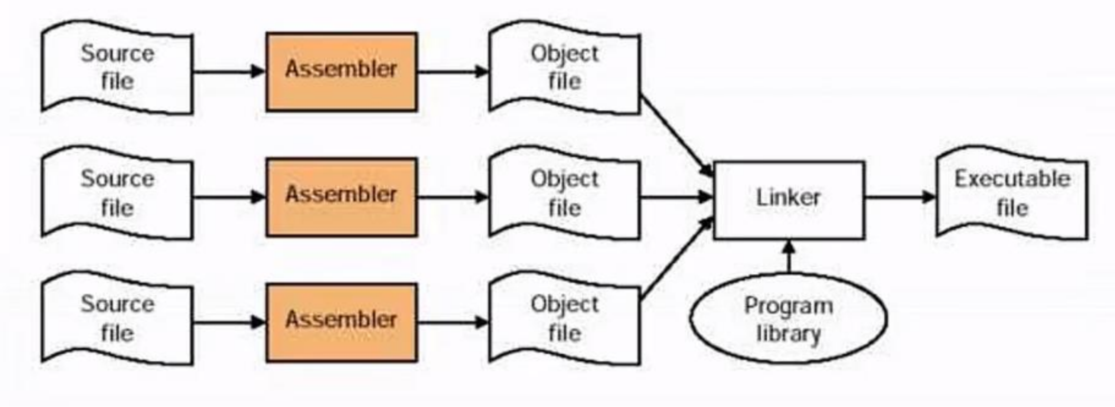


```
NULNULNULEOTNUL*NULNULNULDC3NULNULNULEOTNULNULNULNULNULEOTNULNULNULETXNULEOTNULNULNULEOTNULNULNULETXNULBSNULNULNULFFNULNUL
NULETXNUL.fileNULNULNULNULNULNULNULþÿNULNULgSOHmyprogram.cNULNULNULNULNULNULNULmainNULNULNULNULNULNULNULNULSOHNUL
NULSTXSOHNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL.textNULNULNULNULNULNULNULSOHNULNULNULETXSOH9NULNULNULETXNUL
NULNULNULNULNULNULNULNULNULNULNULNUL.dataNULNULNULNULNULNULNULSTXNULNULNULETXSOHNULNULNULNULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNUL.bssNULNULNULNULNULNULNULNULETXNULNULNULETXSOHNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL.rdataNUL
NULNULNULNULNULEOTNULNULNULETXSOHEOTNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL.xdataNULNULNULNULNULNULENQNULNULNUL
ETXSOHFFNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNULNUL.pdataNULNULNULNULNULNULACKNULNULNULETXSOHFFNULNULNULETXNULNUL
NULNULNULNULNULNULNULNULNULNULNULNULNULNULNULSINULNULNULNULNULNULNULBELNULNULNULETXSOHDC1NULNULNULNULNULNULNULNULNULNULNUL
NULNULNULNULNULNUL__mainNULNULNULNULNULNULNULNUL NULSTXNULprintfNULNULNULNULNULNULNULNUL
NULSTXNULSUBNULNULNUL.rdata$zzzNUL.rdata$zzzNUL
```

Compile, assemble, and link to executable

gcc first.c -o first

- **‘./first’** to execute.



Example

Using GCC.

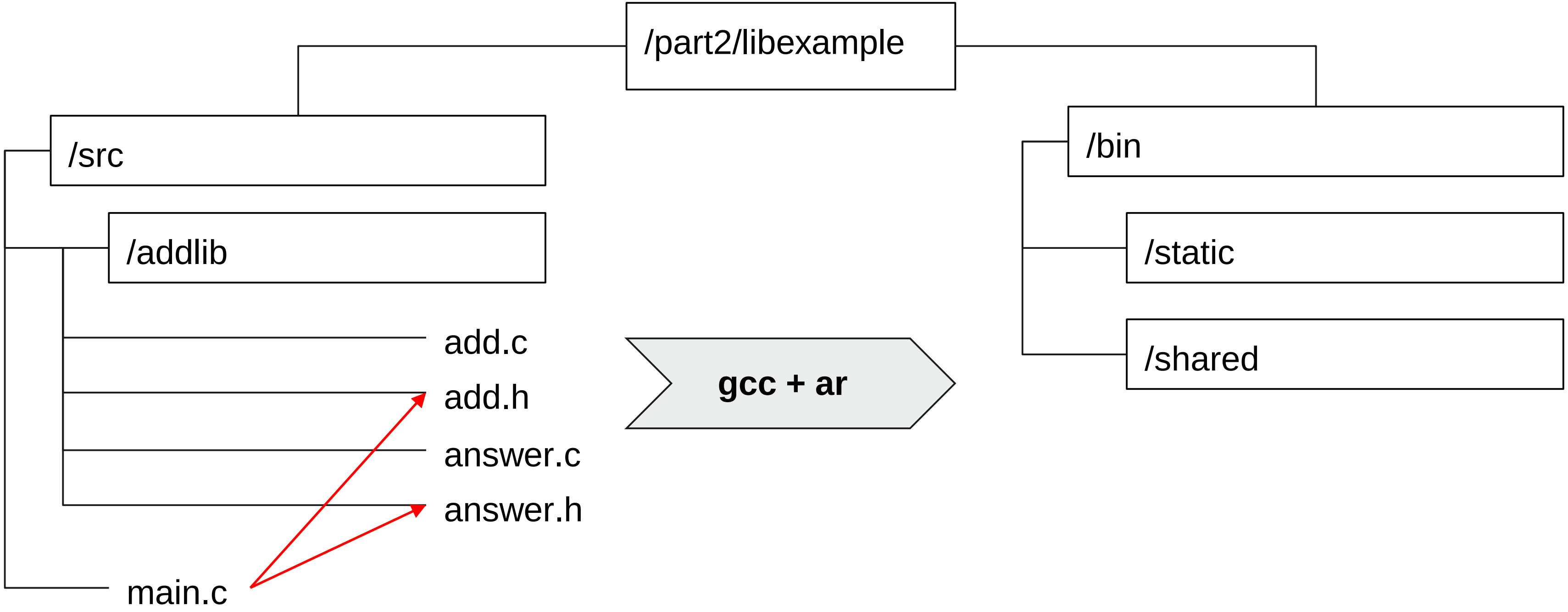
Linux C/C++ Libraries

- Groups together multiple compiled object code files into a single one known as a **library**.
- Some benefits:
  - **Component reuse**: update one library, shared resource takes up less disk space.
  - **Version management**: Linux libraries can cohabitate old and new versions on a single system.
  - **Component specialization**: niche and specialized developers can focus on their core competency on a single library.
- Types:
  - **Static** libraries (.a): Library of object code which is linked with, and becomes part of the application.
  - **Dynamically linked** shared object libraries (.so): Can be dynamically linked at ***run time*** or loaded/unloaded and linked during ***execution***.

GNU archiver (ar)

- **ar** - create, modify, and extract from archives.
- Create library:
  - **ar rcs *<library-name>*.a *<module-1>*.o *<module-2>*.o …**
- List files in library:
  - **ar -t *<library-name>*.a**

Example library structure



Create the object files

- First, we create the object files (in ‘**libexample**’ directory):

  ```
  gcc -c     src/main.c          -o bin/main.o
  ```

- Create the object files for the static library:

  ```
  gcc -c     src/addlib/add.c    -o bin/static/add.o
  gcc -c     src/addlib/answer.c -o bin/static/answer.o
  ```

- Object files for shared libraries need to be compiled as position independent code (-fPIC) because they are mapped to any position in the address space.

  ```
  gcc -c -fPIC src/addlib/add.c    -o bin/shared/add.o
  gcc -c -fPIC src/addlib/answer.c -o bin/shared/answer.o
  ```

Create static library and link statically

- Combine the objects files into a single library/archive:
    - **ar rcs bin/static/libadd.a bin/static/add.o bin/static/answer.o**

- Statically link main.o with the library:

    - **gcc bin/main.o -Lbin/static -ladd -o bin/statically-linked**

- The **-L** flag indicates (non standard) directory where the libraries can be found.

- The **-l** flag indicates the name of the library. Note, that it assumes the library to start with lib and end with .o (so lib and .o must not be specified).

Create shared library and link dynamically

- A shared library is with GCC's **-shared** flag and naming the resultant file with the suffix **.so** rather than **.a**:

  ```
  gcc -shared bin/shared/add.o bin/shared/answer.o -o bin/shared/libadd.so
  ```

- Link dynamically with the shared library (Note: **-ladd-shared** needs to be placed **AFTER main.c**):

  ```
  gcc bin/main.o -Lbin/shared -ladd -o bin/use-shared-library
  ```

- Move the shared library to a default location:

  ```
  sudo mv bin/shared/libadd.so /usr/lib
  ```

  ```
  sudo chmod u=rwx,go=rx /usr/lib/libadd.so
  ```

- Use the shared library with **LD_LIBRARY_PATH**:

  ```
  LD_LIBRARY_PATH=$(pwd)/bin/shared bin/use-shared-library
  ```

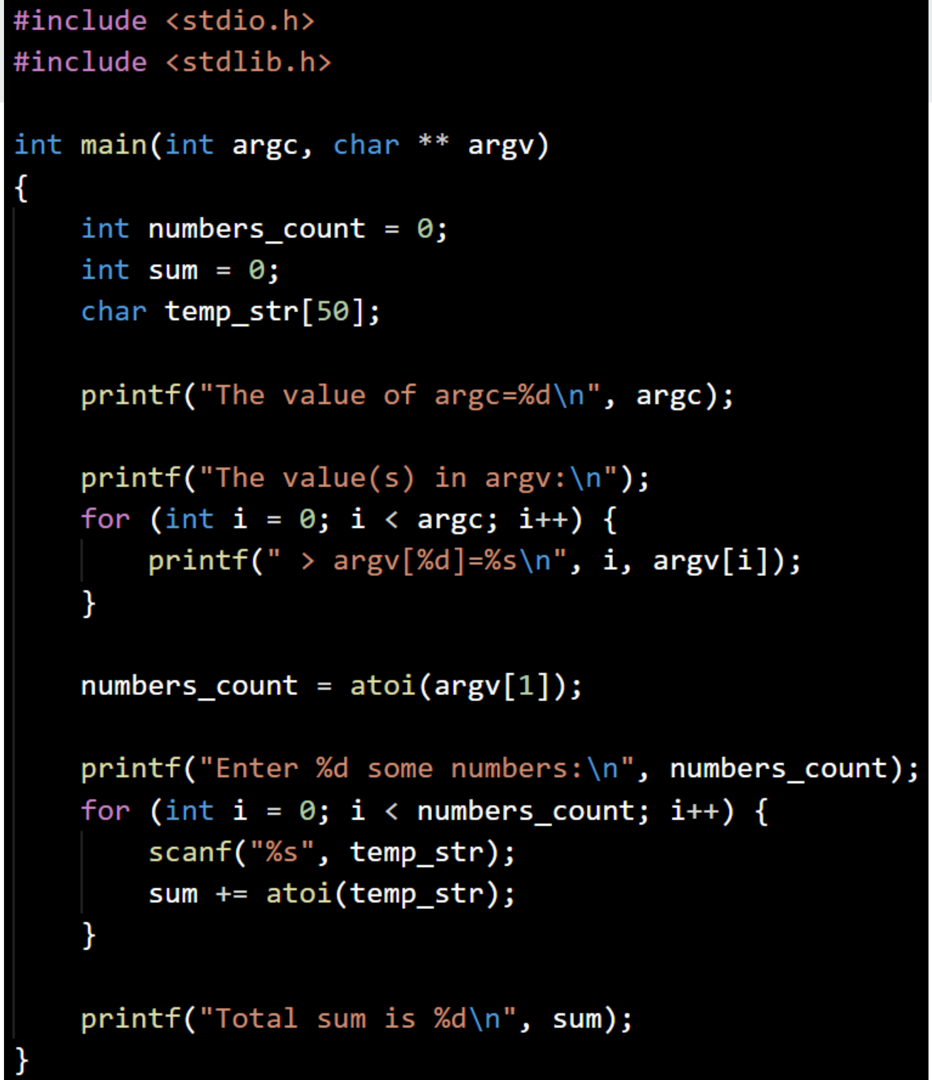
Example

Build a library and use it.

processinp.c

Process input example

```
Initialize variables, sum=0
Print the value of argc
Print the values in argv
num_count= the second value in argv
Do num_count times:
        Read a number from the keyboard and add it to sum
Print the value of sum
```



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

int main(int argc, char ** argv)
{
    int numbers_count = 0;
    int sum = 0;
    char temp_str[50];

    printf("The value of argc=%d\n", argc);

    printf("The value(s) in argv:\n");
    for (int i = 0; i < argc; i++) {
        printf(" > argv[%d]=%s\n", i, argv[i]);
    }

    numbers_count = atoi(argv[1]);

    printf("Enter %d some numbers:\n", numbers_count);
    for (int i = 0; i < numbers_count; i++) {
        scanf("%s", temp_str);
        sum += atoi(temp_str);
    }

    printf("Total sum is %d\n", sum);
}
```

Debugging with GDB

- Compile for debug:
  - **gcc -g <program name>.c**
- Start the GDB environment:
  - **gdb a.out**
- Set a breakpoint, where the program will pause execution:
  - **b [function name, line number]**
- Run the program:
  - **r [command line arguments]**

h	Help
n	Step forward one block of code
s	Step forward one line of code
p [variable]	Print out the value of variable
info locals	Print out the value of all local variables
bt	Show the sequence of the functions called up to this point of execution
q	Quit gdb

Example

Use GDB to debug ‘**processinp.c**’.

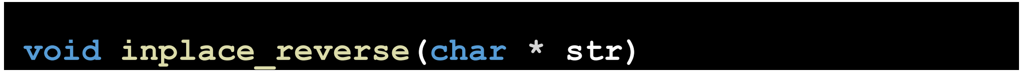
Exercise

Program *MultiAdd*:

Write a program (‘**multiadd.c**’), which takes one or more numbers as arguments, then calculates the total sum and prints the answer to the standard output. Compile the program and execute it. Compile for debug and play with GDB.

Program *ReverseTest*:

Write a program (‘**reversetest.c**’), which reads a string line from the keyboard and prints it backwards. To reverse the input string you have to use the library ‘**reverse**’, which sources are provided in ‘**/day02/reverse/**’ directory. The function you have to use is:



```
void inplace_reverse(char * str)
```

You have all the sources, so you can compile and link the library **statically**. Or you could choose the option to use the **shared** library ‘**reverse**’ in ‘**/usr/lib/libreverse.so**’. The best is if you try **both** approaches.