DAT320
Operating Systems and Systems Programming

A Brief Introduction to C Programming

Today

- History of C
- Hello World
- Toolchain

History

- Initially developed by Dennis Ritchie between 1969 and 1973 at AT&T Bell Labs
- Closely related to development of UNIX
- Inspired by ALGOL

History

- C was derived from B
- B from BCPL
- Rumors say that the name B comes from the name of the inventors wife, Barbara

First program

```
int main( void )
{
     return 0;
}
```

$gcc first.c –o first

$ ./first

$

First program

- Every full C program begins inside a function called "main“

Parts of a C-program

```
int main( void )
{
    return 0;
}
```

Hello world

```c
#include <stdio.h>

int main(void)
{
        /* The hello world program*/
    printf("hello, world\n");
}
```

user@badne7:~$ gcc hello.c -o hello

user@badne7:~$ ./hello

hello, world

Summary of Hello, World!

- The first line of the program *#include* <*stdio.h*> is a preprocessor command
    - Tells the C compiler to include the *stdio.h* file in-place before actual compilation
- *int main(void)* is the main function where program execution begins.
- /*...*/ is a simple comment and will be ignored by the compiler.
- *printf(...)* is another C function, which causes the message "Hello, World!" to be displayed on the screen.
- *return* 0; terminates the main() function and returns the value 0 to the OS (shell).

The Preprocessor Directive:

- Executed **before** compilation
- Include other files (in-place):
  #include<…>
  #include "…."
- Simple macros:
  #define BUFFER_SIZE 100
- Simple control structures:
  #if
  #endif

Functions in C

```
return_type function_name( parameter list )
{
        body of the function
}
```

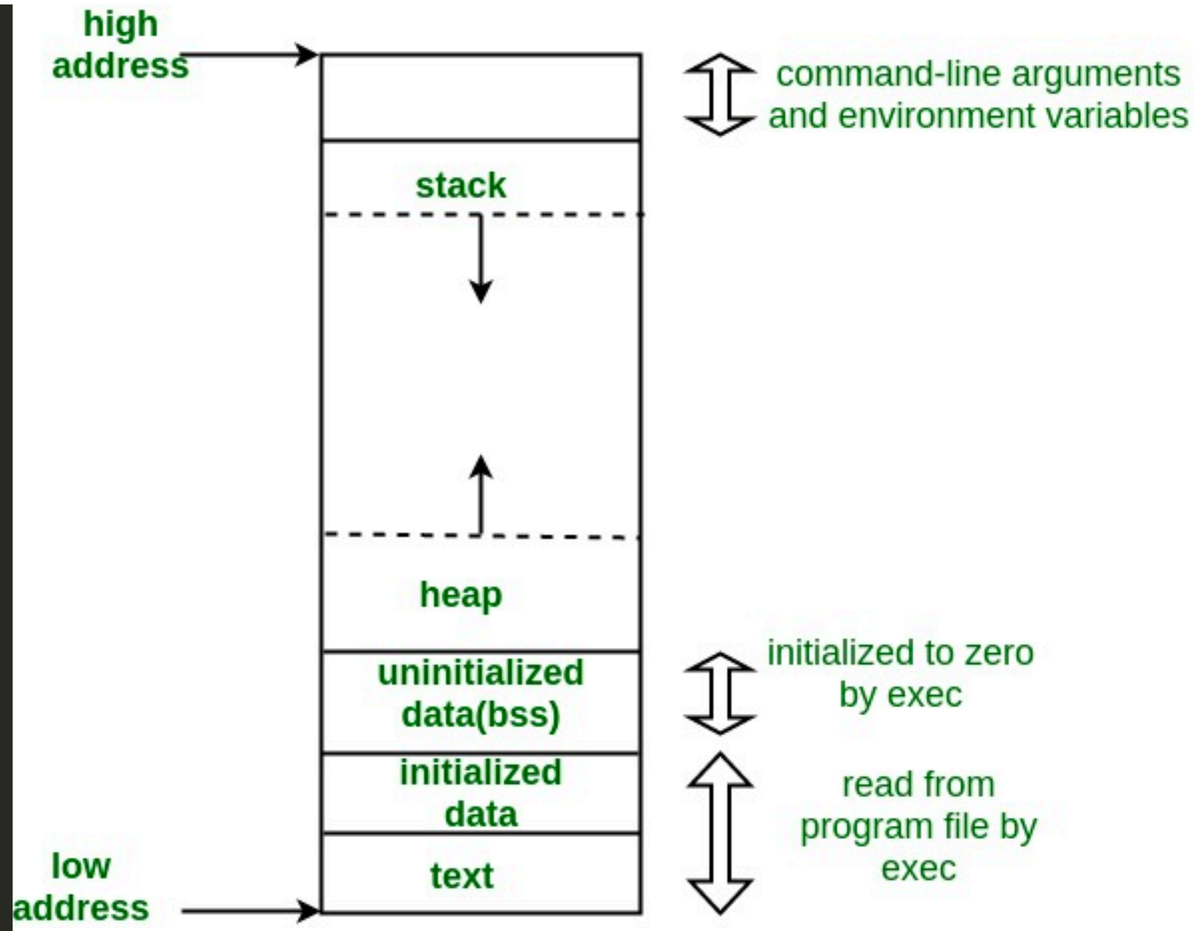
Functions in C

```c
#include <stdio.h>
int my_add(int, int);
int main(void)
{
    int sum = 0;
    sum = my_add( 3,5);
    printf("hello, world\n");
    printf("%d + %d = %d\n", 3,5, sum );
}
int my_add( int a, int b)
{
        return a+b;
}
```

Structure of a C program

```c
#include <stdio.h>
#include <stdlib.h>
void f();
int main()
{
    static int x = 0;
    int a = 10;
    int* b = (int *)malloc(sizeof(int));
    *b = 100;
    printf("%d, %d, %d, %p\n", x, a, *b, b);
    f();
    return 0;
}

void f()
{
    printf("hello!\n");
    f();
    return;
}
```



C Tutorial

http://www.tutorialspoint.com/cprogramming/c_quick_guide.htm

Make: Automate Building Software

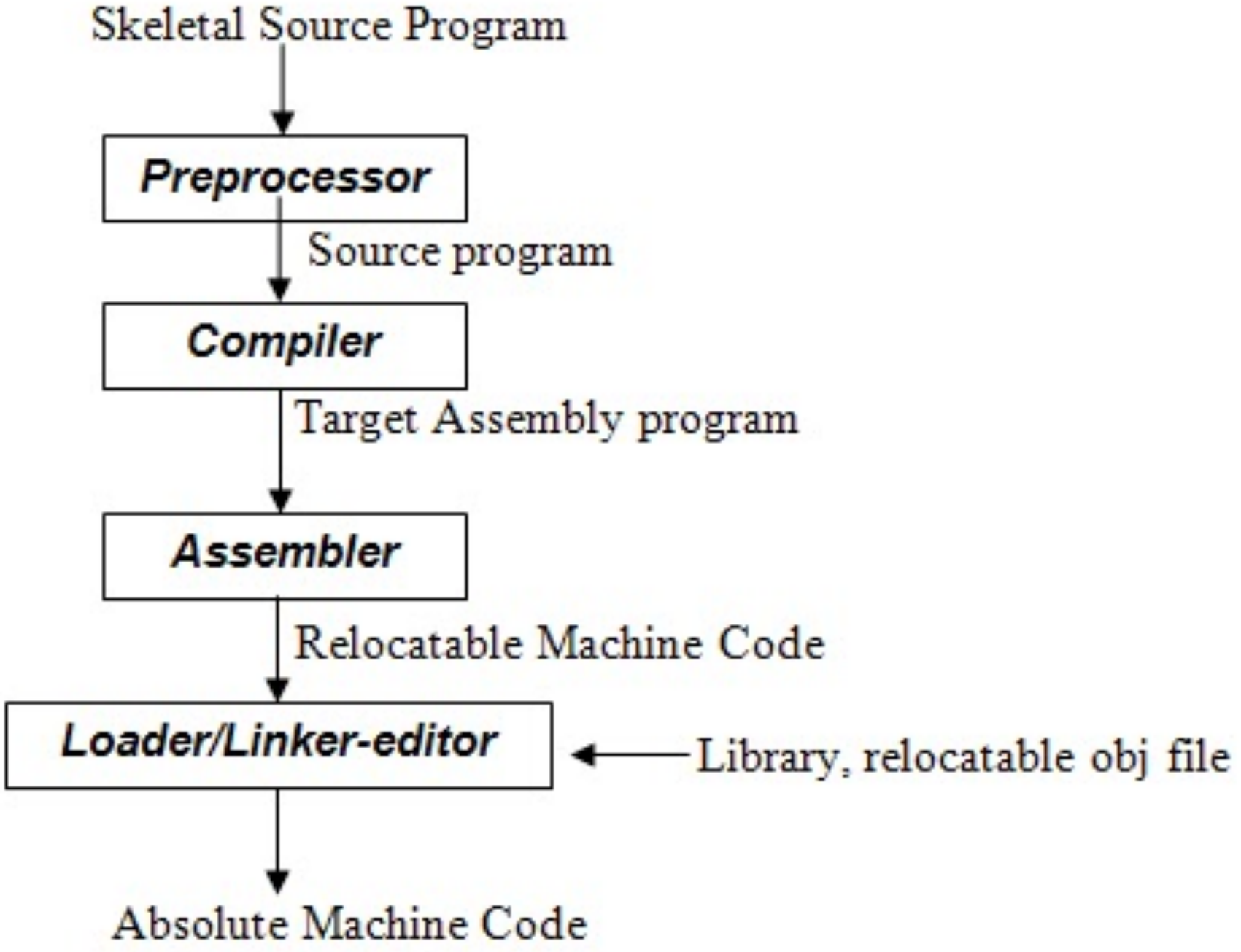
target: dependencies
[tab] system command

Make

Makefile:

```
hello: hello.c
	gcc hello.c -o hello
```

```
user@badne7:~$ make
gcc hello.c -o hello
user@badne7:~$
user@badne7:~$ make
make: `hello' is up to date.
user@badne7:~$
```


Skeletal Source Program
Preprocessor
Source program
Compiler
Target Assembly program
Assembler
Relocatable Machine Code
Loader/Linker-editor
Library, relocatable obj file
Absolute Machine Code

Q1

- How do you use gcc to only produce the .o file?
- What is the difference between generating only the .o file, and building the hello executable done in the previous compilation above?

```
user@badne7:~$ gcc -help
Usage: gcc [options] file...
Options:
  -pass-exit-codes          Exit with highest error code from a phase
  --help                    Display this information
    .
    .
 -c                        Compile and assemble, but do not link

user@badne7:~$ gcc -c hello.c -o hello.o
```

Q2 and Q3

Give the command for compiling with debug enabled instead of normal compilation for the two examples shown in Listing 2 and Listing 3. Explain how to turn debugging on/off for the two cases.

Give a brief pros and cons discussion for the two methods to add debug code shown in Listing 2 and Listing 3.

gcc -D defines a macro to be used by the preprocessor:

```
gcc -DDEBUG myfile.c -o myfile
```

```
user@badne7:~$ gcc  hello.c -o hello
user@badne7:~$ ./hello
3 + 5 = 8
user@badne7:~$ gcc  -DMYSYMBOL hello.c -o hello
user@badne7:~$ ./hello
hello, world
3 + 5 = 8
```

```c
#include <stdio.h>
int main(void)
{
#ifdef MYSYMBOL
    printf("hello, world\n");
    #endif
     printf("%d + %d = %d\n", 3,5,my_add(3,5) );
}
int my_add( int a, int b)
{
        return a+b;
}
```

- .data - **initialized data**.
- .bss - uninitalized data.
- .rodata – read only

```
#include <stdio.h>
int myvar1=0;
int myvar2=1;
const int myvar3=1;
int main(void)
{
    …
```

float vs double multiply

```
user@badne7:~$ gcc -help
Usage: gcc [options] file...
Options:
  -pass-exit-codes          Exit with highest error code from a phase
  --help                    Display this information
   .
   .
  -S                        Compile only; do not assemble or link
```

```
float mult(float a, float b)
{
        return a*b;
}

double multd(double a, double b)
{
        return a*b;
}
```

```
user@badne7:~$ gcc hello.c -S -o hello.asm
```

Hint: look at hello.asm. Search for ‘mult’ and ‘multd’