YORK UNIVERSITÉ UNIVERSITY

Introduction to C

EECS 2031

Gias Uddin
guddin@yorku.ca
https://giasuddin.ca

Acknowledgement

- Some of the covered materials are based on previous EECS2031 offerings:
  - Song Wang, Uyen Trang (UT) Nguyen, Pooja Vashisth, Hui Wang, Manos Papagelis

Origins of C

- K&R C:
  - **C was developed at Bell Laboratories** by mainly Ken Thompson & Dennis Ritchie
  - Brian **K**ernighan and Dennis **R**itchie wrote *The C Programming Language (1978)*

- C89/C90:
  - ANSI standard X3.159-1989 (completed in 1988; formally approved in December 1989)
  - International standard ISO/IEC 9899:1990

- C99:
  - International standard ISO/IEC 9899:1999
  - Incorporates changes from Amendment 1 (1995)

Applications

C programming language is widely used in various applications due to its **efficiency**, **portability**, and **low-level control**.

1. **Operating Systems Development**: C is often used in the development of operating systems like **Unix, Linux, and Windows**. Its low-level memory manipulation capabilities make it suitable for managing system resources.
2. **Embedded Systems**: C is the primary language for programming embedded systems, such as **microcontrollers** and **drone controllers**, where efficiency and low-level control are critical.
3. **System Software**: C is used to develop system software components like **device drivers**, **file systems**, and **system utilities**.
4. **Compilers and Interpreters**: C is often used to implement **compilers** and **interpreters** for other programming languages. The C language itself is typically compiled.
5. **Game Development**: C and C++ are common choices for **game development** due to their performance and the ability to access hardware directly.

C basics

- **The first program –what it looks like**
- Compile and run C program
- Basic syntax
  - Comments
  - Variables
  - Functions
  - Basic IO functions
  - Expression
  - Statements
  - Preprocessing: #include, #define
  - …

The first program –what it looks like

```c
#include <stdio.h>
/* import standard io header */

/* salute the world */

int main ()
{
   printf( "Hi, world\n" );
   return 0;
}
```

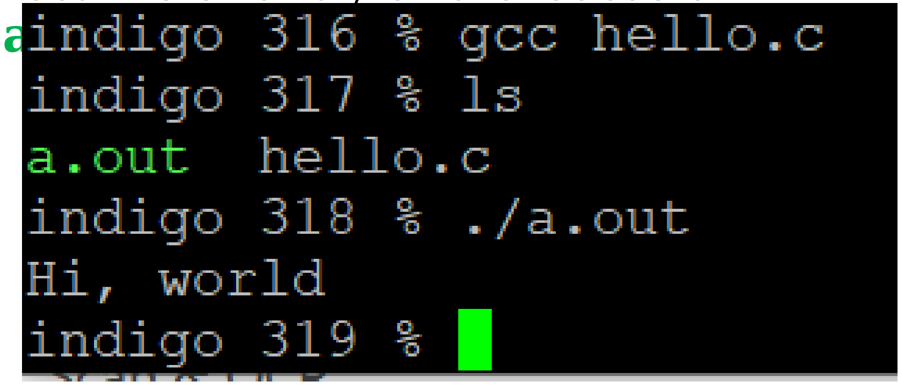
```c
#include <stdio.h>
/* import standard io header */

/* salute the world */

int main (int argc, char** argv)
{
   printf( "Hi, world\n" );
   return 0;
}
```

hello.c, first.c, any_name.c

Compiling and running a C program

- C programs (source code) are in files ending with **.c** e.g., **hello.c**

- To compile a C program, naturally in Unix, we use **gcc** to compile c:
  - **`% gcc hello.c`**
  - If no syntax error, complier returns silently and creates an executable program named **a**

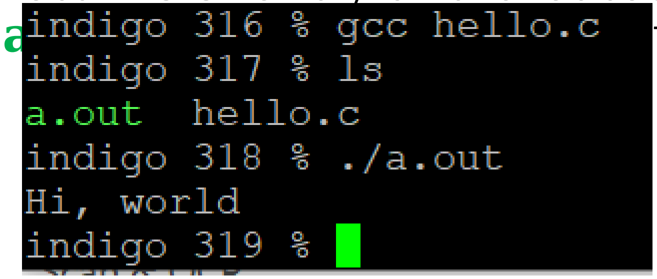


```
indigo 316 % gcc hello.c
indigo 317 % ls
a.out   hello.c
indigo 318 % ./a.out
Hi, world
indigo 319 %
```

Compiling and running a C program

- C programs (source code) are in files ending with **.c** e.g., **hello.c**

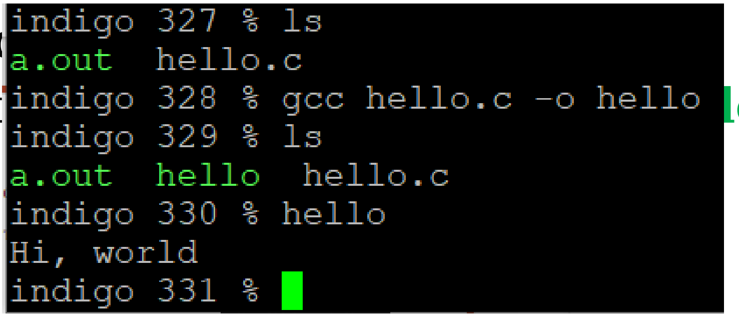
- To compile a C program, naturally in Unix, we use **gcc** to compile c:
  - **`% gcc hello.c`**
  - If no syntax error, complier returns silently and creates an executable program named a tory)



```
indigo 316 % gcc hello.c
indigo 317 % ls
a.out   hello.c
indigo 318 % ./a.out
Hi, world
indigo 319 %
```

- To run
  - % **./a.out** or **a.out**

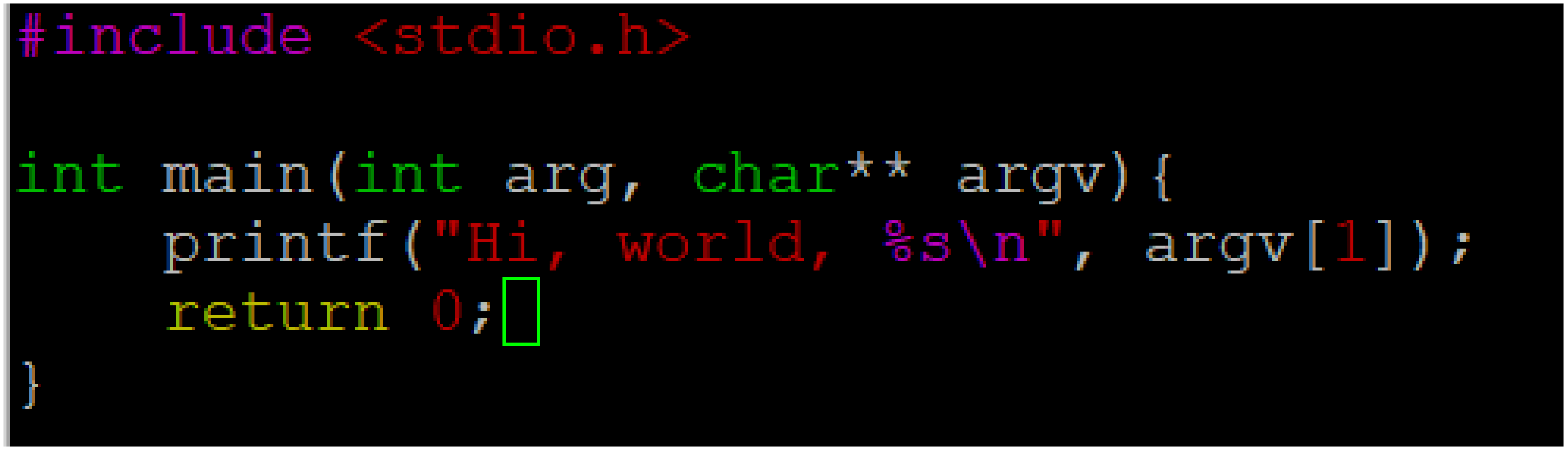
  - **`% gcc hell`**
  - create an ex lo (in the current directory)



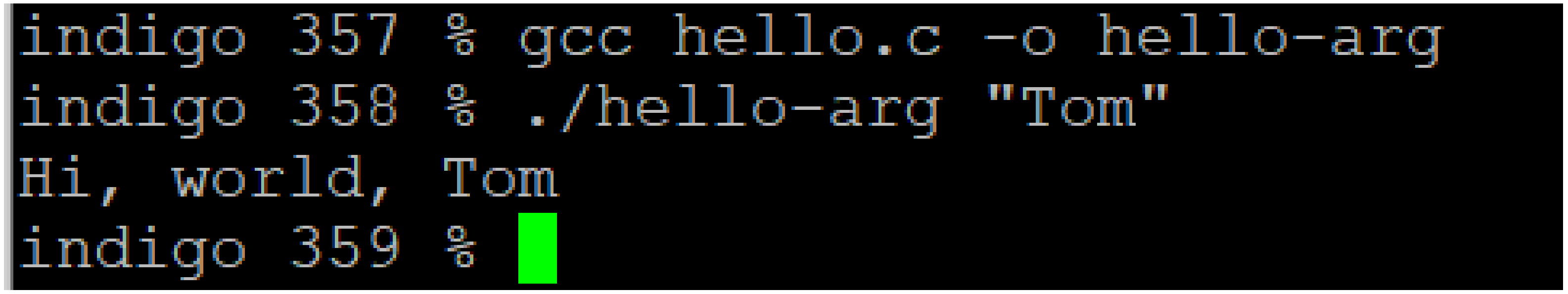
```
indigo 327 % ls
a.out   hello.c
indigo 328 % gcc hello.c -o hello
indigo 329 % ls
a.out   hello   hello.c
indigo 330 % hello
Hi, world
indigo 331 %
```

Compiling and running a C program

- C program with arguments



```
#include <stdio.h>

int main(int arg, char** argv){
    printf("Hi, world, %s\n", argv[1]);
    return 0;
}
```



```
indigo 357 % gcc hello.c -o hello-arg
indigo 358 % ./hello-arg "Tom"
Hi, world, Tom
indigo 359 %
```

GNU Compiler Collection (gcc)

- GCC is a set of compilers for **various languages**. It provides all of the infrastructure for building software in those languages from source code to assembly.

- The compiler can handle compiling everything on its own, but you can use various flags to breakdown the compilation steps.

- **Default**: C89/90 + some C99 features

- Example:
  - gcc [flags] [infile(s)]
  - To compile using C99: `gcc -std=c99 hello.c`

Common GCC Flags

-o [EXECUTABLE NAME] : names executable file

-O*x* : Code optimization

- **-O0** : Compile as fast as possible, don’t optimize [this is the default]
- **-O1**, **-O2**, **-O3**: Optimize for reduced execution time [higher numbers are more optimized]
- **-Os** : Optimize for code size instead of execution time.
- **-Og** : Optimize for execution time, but try to avoid making interactive debugging harder.

-g : produce “debug info”: annotate assembly so gdb can find variables and source code

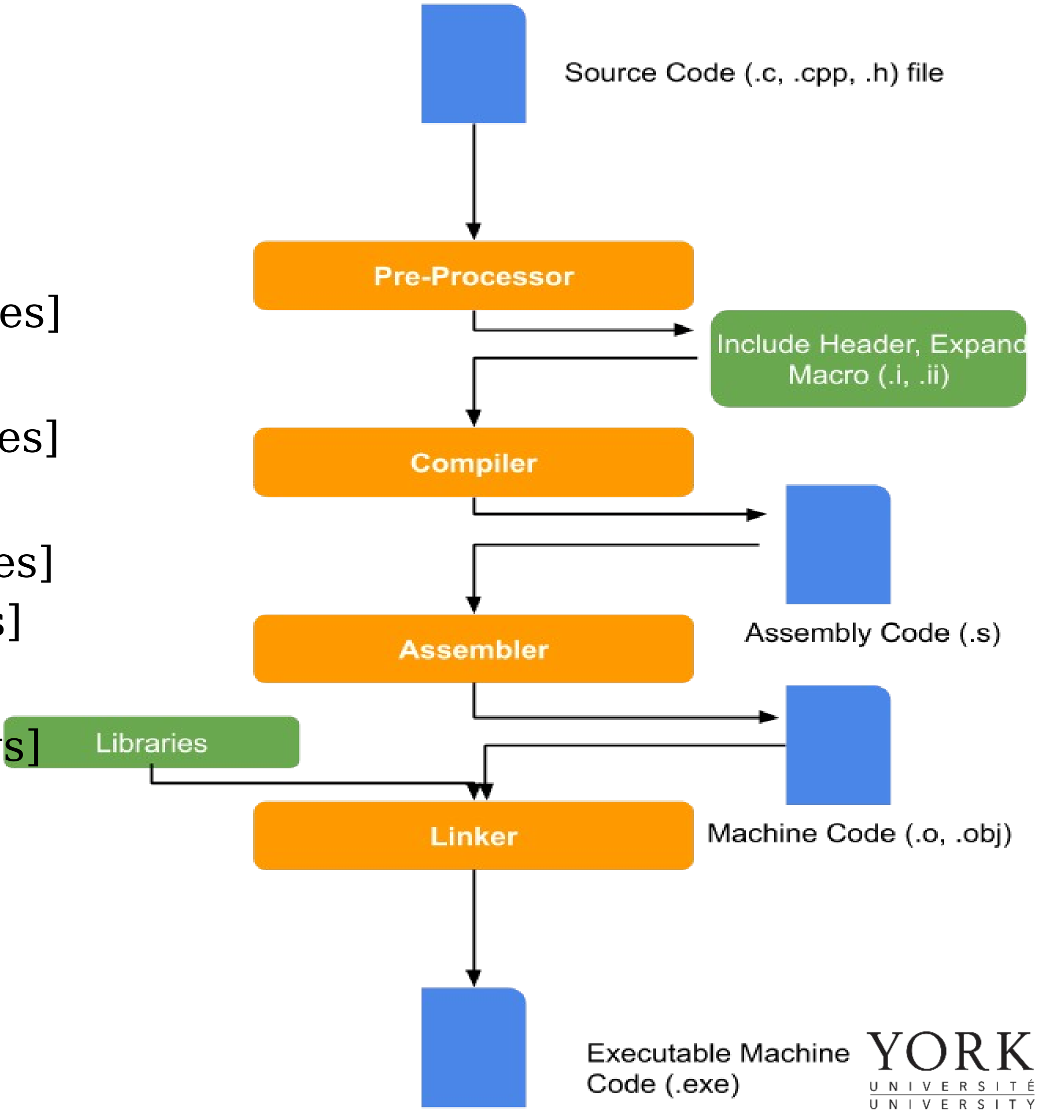
-Wall : enable many “warning” messages that *should* be on by default

-Werror : turns all warnings into errors

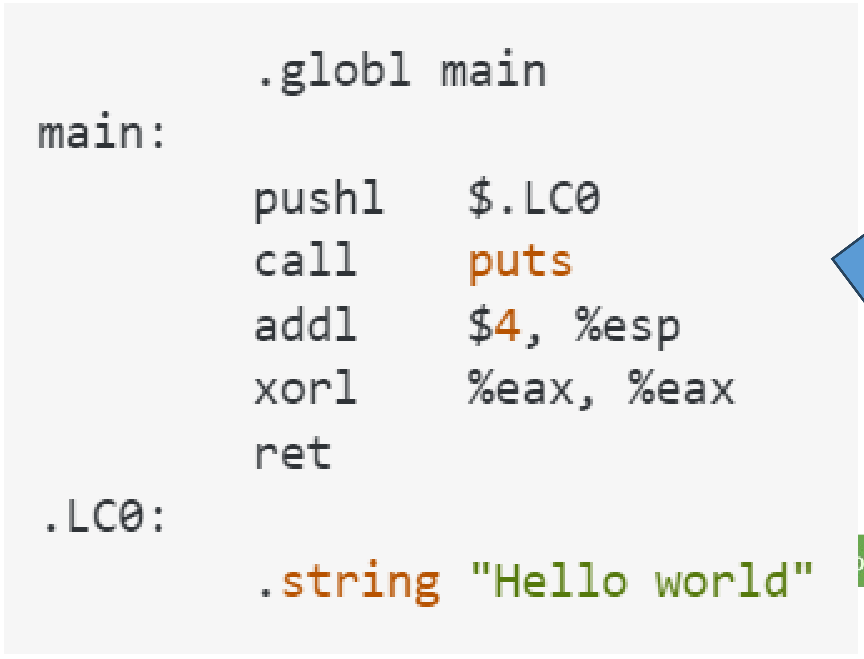
-std=c99 : use the 1999 version of the C standard and disable some (not all!) extensions

Compilation: transformation of program code to machine understandable code

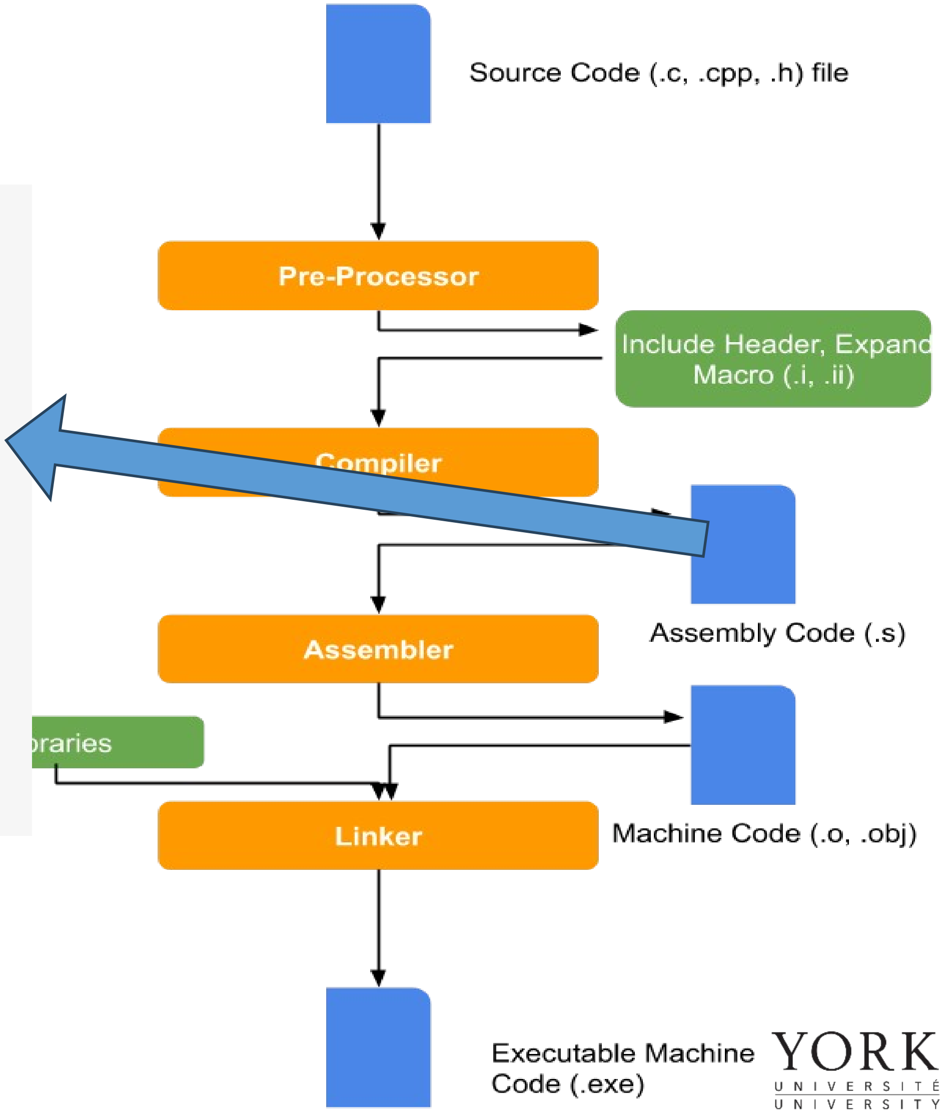
- Pre-Processor
  - $ gcc -E [flags] [filenames]
- Compiler
  - $ gcc -S [flags] [filenames]
- Assembler
  - $ gcc -c [flags] [filenames]
  - $ objdump -d [filenames]
- Linker
  - $ gcc -o [exename] [flags] [filenames]


Source Code (.c, .cpp, .h) file
Pre-Processor
Include Header, Expand Macro (.i, .ii)
Compiler
Assembly Code (.s)
Assembler
Libraries
Machine Code (.o, .obj)
Linker
Executable Machine Code (.exe)

Compilation: transformation of program code to machine understandable code



```
        .globl main
main:
        pushl   $.LC0
        call    puts
        addl    $4, %esp
        xorl    %eax, %eax
        ret
.LC0:
        .string "Hello world"
```



Source Code (.c, .cpp, .h) file
Pre-Processor
Include Header, Expand Macro (.i, .ii)
Compiler
Assembly Code (.s)
Assembler
raries
Linker
Machine Code (.o, .obj)
Executable Machine Code (.exe)

C basics

- The first program –what it looks like
- Compile and run C program
- **Basic syntax**
  - Comments
  - Variables
  - Functions
  - Basic IO functions
  - Expression
  - Statements
  - Preprocessing: #include, #define
  - …

Comments

- ANSI-C (C89) /* comment */
- •Span multiple lines /* .....

  .....*/
- May not be nested /* /* */ */
- Good practice to comment things. But don’t write trivial ones

- C99 feature // ("single-line" comment)

gcc hello.c –default C89 + some C99.

C variables

- Store data, whose value can change.
  - Declaration and initialization.
  - **int x;**
  - **int x =5;**

- Variable names
  - combinations of letters (including underscore character _), and numbers.
  - that do not start with a number; avoid starting with _;
  - are not a keyword.
  - uppercase and lowercase letters are distinct (x ≠ X).

C variables

- Store data, whose value can change.
  - Declaration and initialization.
  - **int x;**
  - **int x =5;**

- Variable names
  - combinations of letters (including underscore character _), and numbers.
  - that do not start with a number; avoid starting with _;
  - are not a keyword.
  - uppercase and lowercase letters are distinct (x ≠ X).

- Examples: Identify valid and invalid variable names
  - **`abc, aBc, abc5, aA3_ , my_index`**
  - **`5sda, _360degrees, _temp, char, struct, while`**

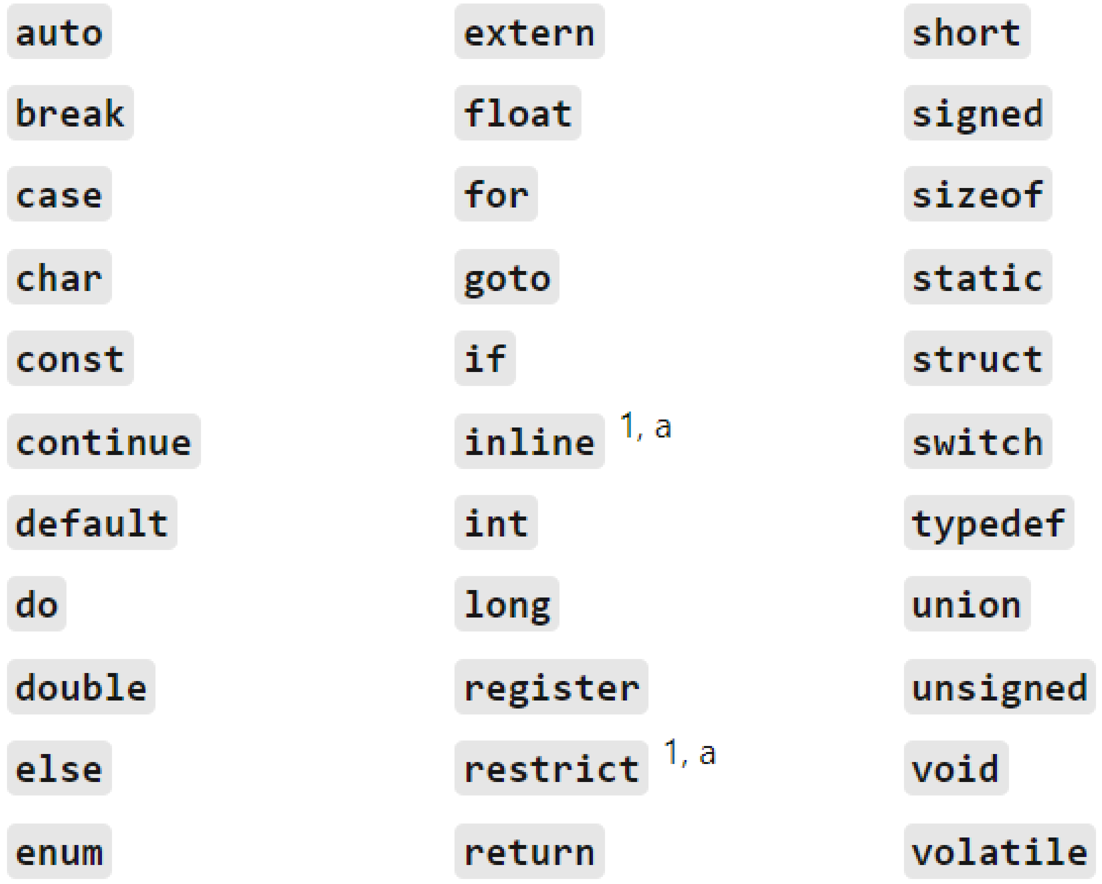
C variables (keyword)

char--**characters**

int--**integers**

float **--single precision floating point numbers**

double--**double precision floating point**



`auto`	`extern`	`short`
`break`	`float`	`signed`
`case`	`for`	`sizeof`
`char`	`goto`	`static`
`const`	`if`	`struct`
`continue`	`inline` [1, a]	`switch`
`default`	`int`	`typedef`
`do`	`long`	`union`
`double`	`register`	`unsigned`
`else`	`restrict` [1, a]	`void`
`enum`	`return`	`volatile`

functions

```
return_type functionName(parameter type name, ……)
{body block}
```

functions

```
return_type functionName(parameter type name, ……)
{body block}


   int main(){…}

   int sum (int i, int j){
       int s = i+ j;
       return s;               /*  return i+j;  */
   }

   void display (double i){
           printf("this is %f", i);
   }
```

functions

```
#include <stdio.h>

/* function definition*/
int sum (int i, int j){
        return i+j;
}
int main()
{
    int x = 2, y = 3;
    int su= sum(x , y);
    printf("Sum is %d\n", su);
}
```

```
/* Contains declaration
(prototype) of printf() */
```

Point of function call

Point of function call (from stdio.h)

functions

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

/* Contains declaration (prototype) of printf() */

```
int main()
{
    int x = 2, y = 3;
    int su= sum(x , y);
    printf("Sum is %d\n", su);
}
```

Not Defined or Declared before function call

Point of function call ✗

Point of function call (from stdio.h)

```
/* function definition*/
int sum (int i, int j){
        return i+j;
}
```

functions

```
#include <stdio.h>


/* function declaration*/
int sum(int, int);     /* intsum(inta, intb) */

int main(){

    int x = 2, y = 3;

    int su= sum(x , y);

    printf("Sum is %d\n", su);

}

/* function definition*/

int sum (int i, int j){

       return i+j;

}
```

Declared before function call

Basic I/O functions
<stdio.h>

- Every program has a Standard Input: keyboard
- Every program has a Standard Output: console/terminal/screen …
- Can be redirected in Unix
  - **< inputFile**
  - **> outputFile**

- **int printf(char *format, arg1, ….);** Formats and prints arguments on standard output
  - **printf("This is a test %d \n", x)**

- **int scanf(char *format, arg1, ….);** Formatted input from standard input
  - **scanf("%d %d", &x, &y)**

- **int getchar();** Reads and returns the next char on standard input

- **int putchar(int c)** Writes the character c on standard output

printf

format string

/* conversion specification */

- `printf("This is day %d of Sep\n", x)`
  - Formats and prints arguments on standard output [illegible]
  - Returns number of chars printed (often discarded)

- Format string contains: 1) regular chars 2) conversion specifications
  - %d **to be replaced/filled with an integer – decimal** “place holders”
  - %c **to be replaced/filled with a character**
  - %f **to be replaced/filled with a floating point number (float, double)**
  - %s **to be replaced/filled with a "string" (array of chars)**
  - ...

```
printf("Hello World\n");              Hello World
printf("%s\n", "Hello World");        Hello World
printf("%s World\n", "Hello");        Hello World
```

```
int a = 15; int b = 3;
printf("This is day " + a + " of Jan.\n");          This is day 15 of Jan.

printf("This is day " + a + ", week " + b + "of Jan.\n");      ✗
                                                This is day 15, week 3 of Jan.
```

functions

```
#include <stdio.h>


/* function declaration*/
Int sum(int, int);    /* intsum(inta, intb) */

int main(){

   int x = 2, y = 3;

   int su= sum(x , y);

   printf("Sum of %d and %d is %d\n", x, y, su);

   }

/* function definition*/

int sum (int i, int j){

      return i+j;

}
```

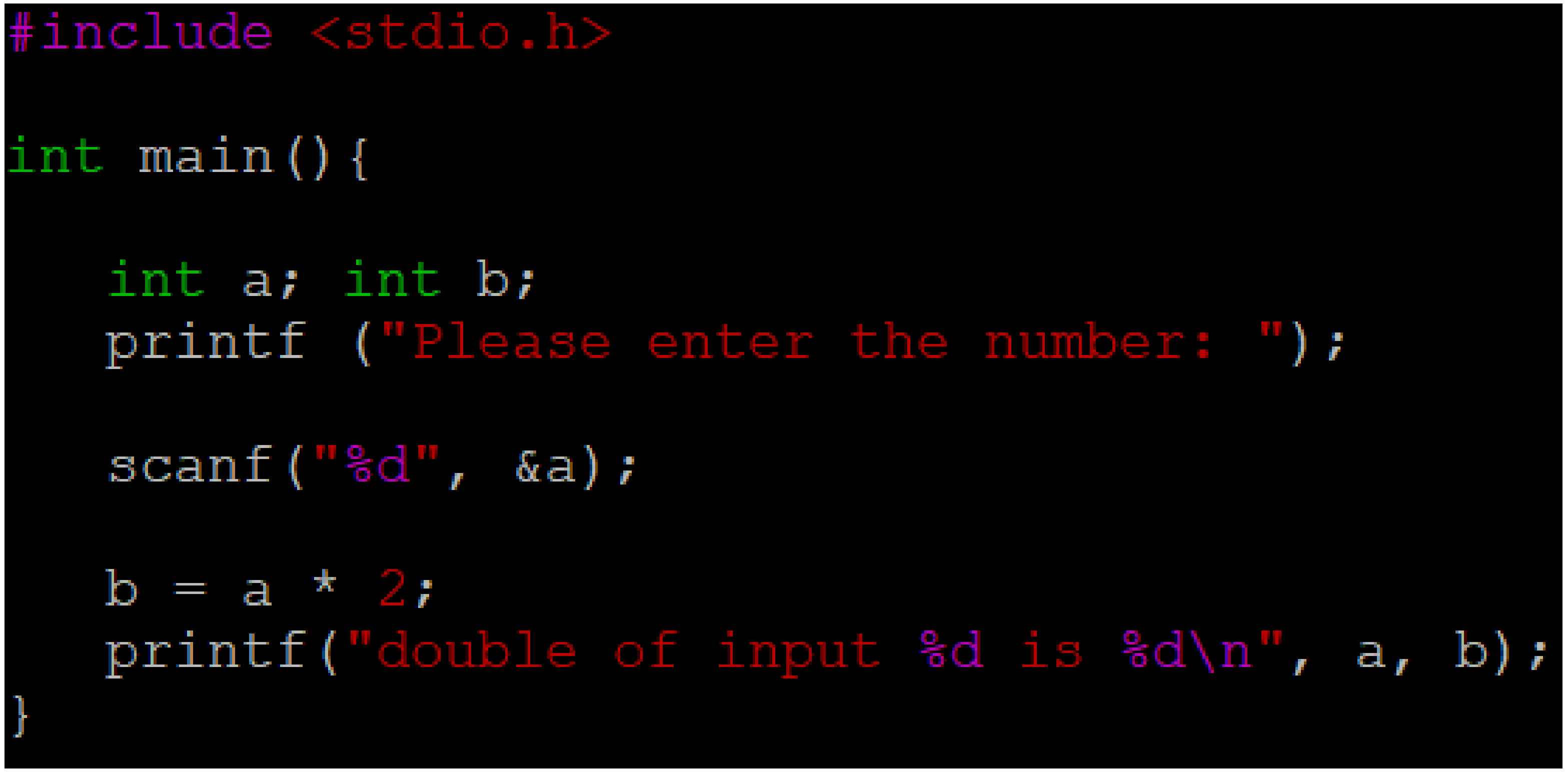
scanf()

- **int x;**
- **•scanf("%d", &x)**
  - opposite to **printf()**
  - formatted input from standard input
  - return number of successful scans/conversions (usually discarded) or EOF
  - Wait for standard input, then converts input to int, and assign value to **x**

- Format string contains: 1) regular chars 2) conversion specifications
  - **%d convert input to an integer –decimal**
  - **%c convert input to a character**
  - **%f convert input to a floating-point number (%lf for double)**
  - **%s convert input to a "string"**

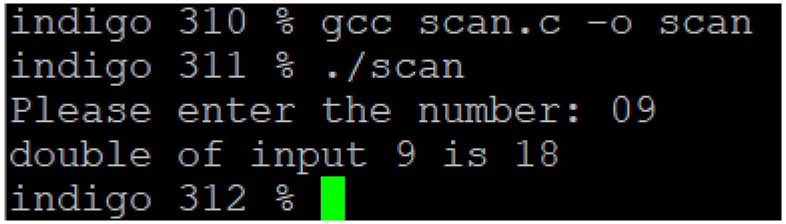
- **&x** -> memory address of **x**.

scanf() example I



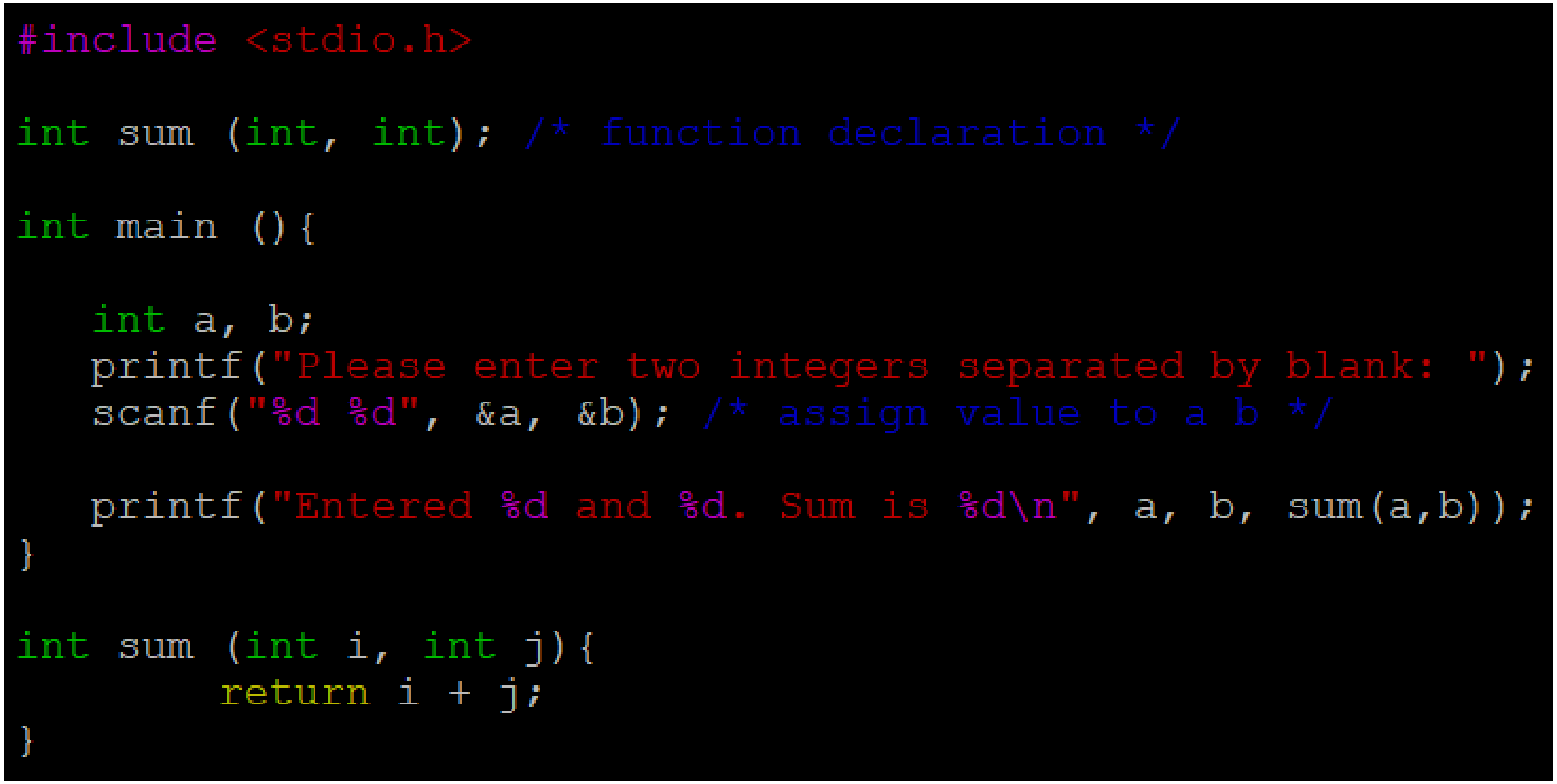
```c
#include <stdio.h>

int main(){

   int a; int b;
   printf ("Please enter the number: ");

   scanf("%d", &a);

   b = a * 2;
   printf("double of input %d is %d\n", a, b);
}
```

- **&a** memory address of **a**.   Details later. Take as it is

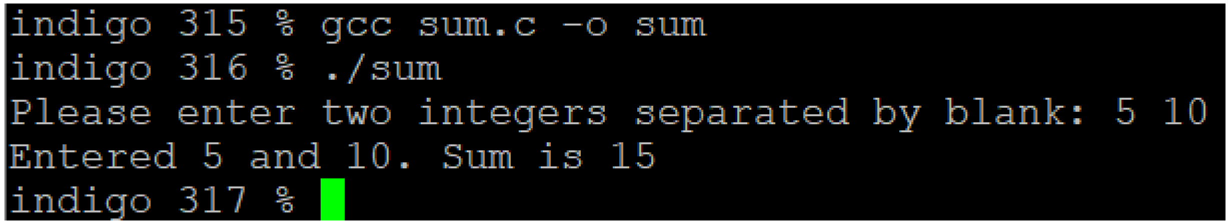


```
indigo 310 % gcc scan.c -o scan
indigo 311 % ./scan
Please enter the number: 09
double of input 9 is 18
indigo 312 % 
```

scanf() example II



```c
#include <stdio.h>

int sum (int, int); /* function declaration */

int main (){

   int a, b;
   printf("Please enter two integers separated by blank: ");
   scanf("%d %d", &a, &b); /* assign value to a b */

   printf("Entered %d and %d. Sum is %d\n", a, b, sum(a,b));
}

int sum (int i, int j){
        return i + j;
}
```



```
indigo 315 % gcc sum.c -o sum
indigo 316 % ./sum
Please enter two integers separated by blank: 5 10
Entered 5 and 10. Sum is 15
indigo 317 %
```