COMSW 1003-1

Introduction to Computer Programming in C

Lecture 3

Spring 2011

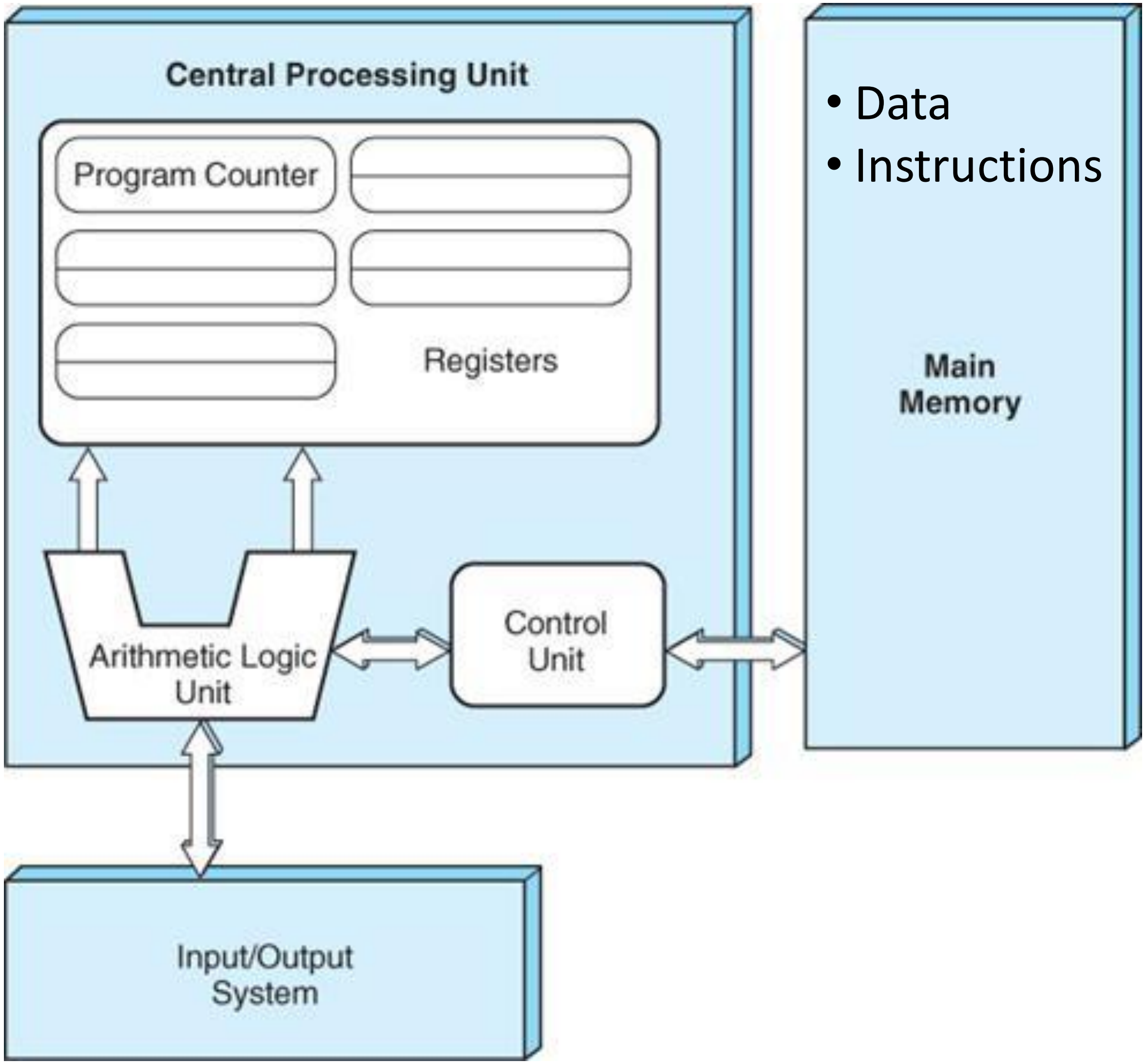
Instructor: Michele Merler

http://www1.cs.columbia.edu/~mmerler/comsw1003-1.html

Today

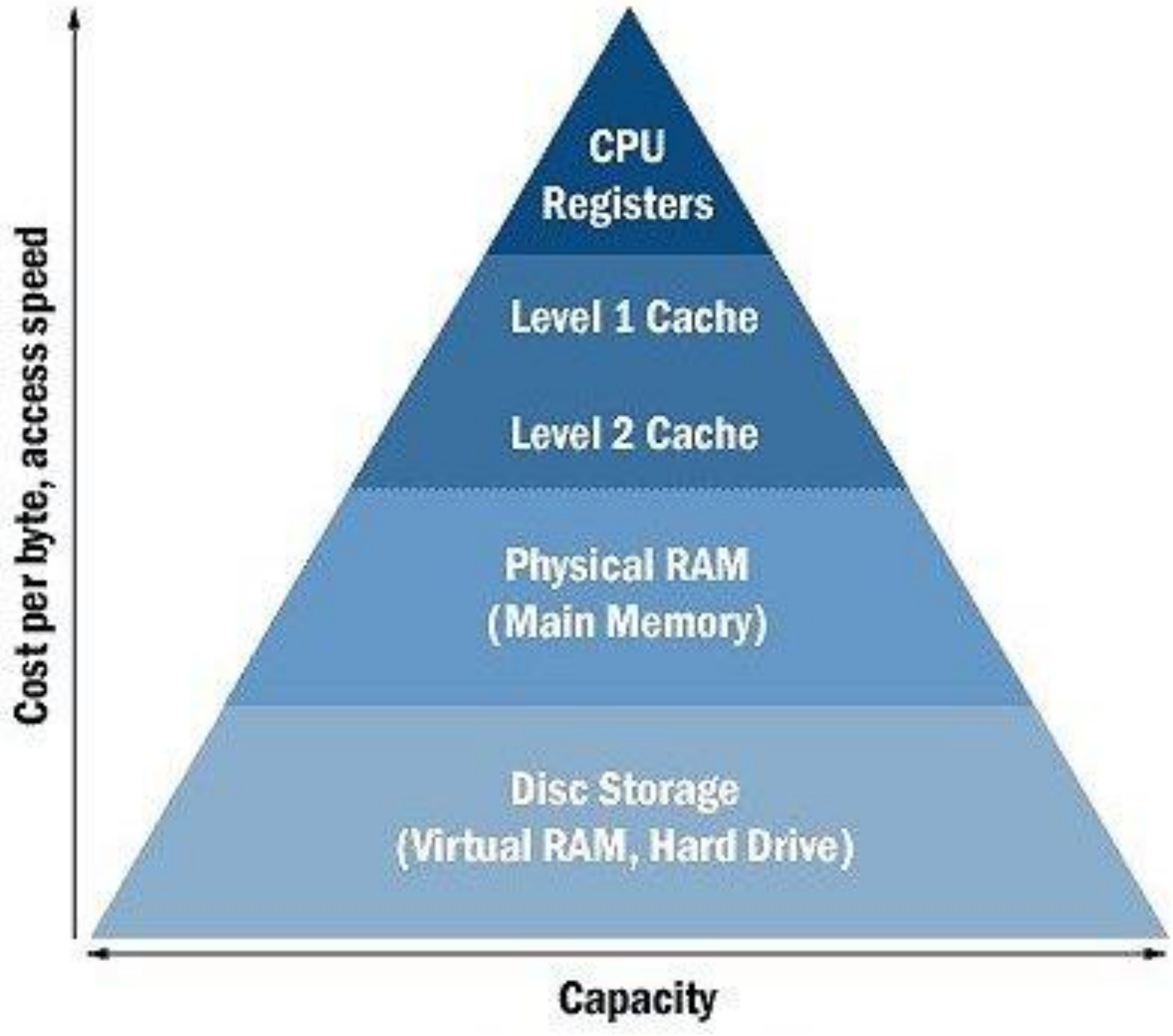
- Computer Architecture (Brief Overview)
- “Hello World” in detail
- C Syntax
- Variables and Types
- Operators
- printf (if there is time)

Von Neumann Architecture



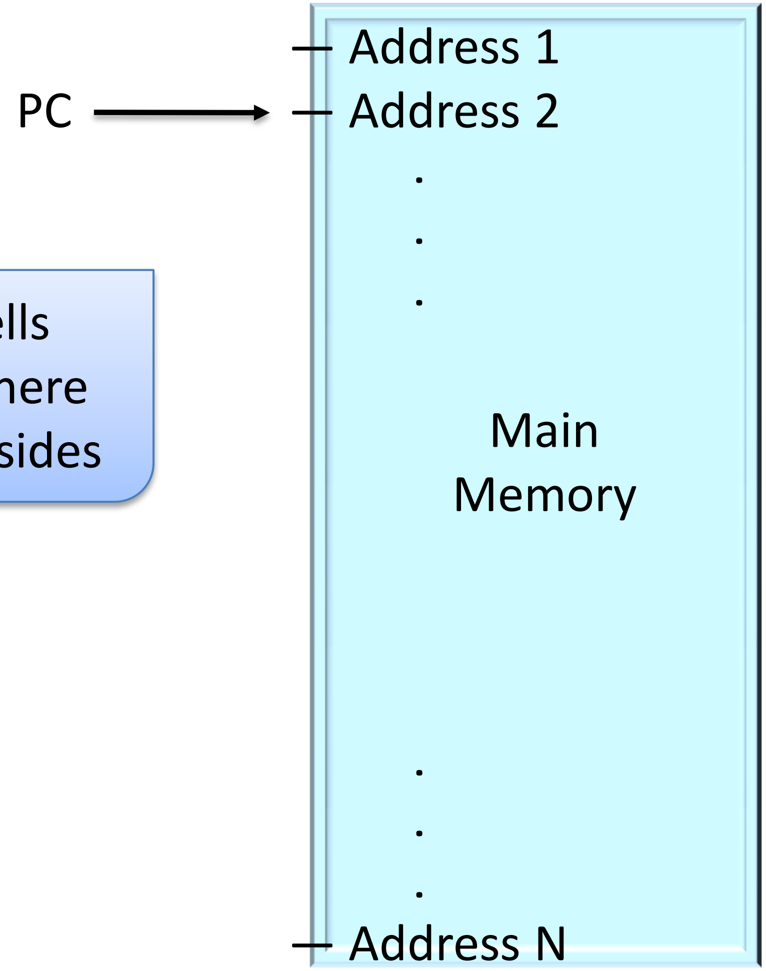
- Data
- Instructions

Computer Memory Architecture


CPU Registers
Level 1 Cache
Level 2 Cache
Physical RAM (Main Memory)
Disc Storage (Virtual RAM, Hard Drive)
Cost per byte, access speed
Capacity

Von Neumann Architecture

The Program Counter (PC) **points** (= tells the CPU) to the address in memory where the next instruction to be executed resides



Von Neumann Architecture

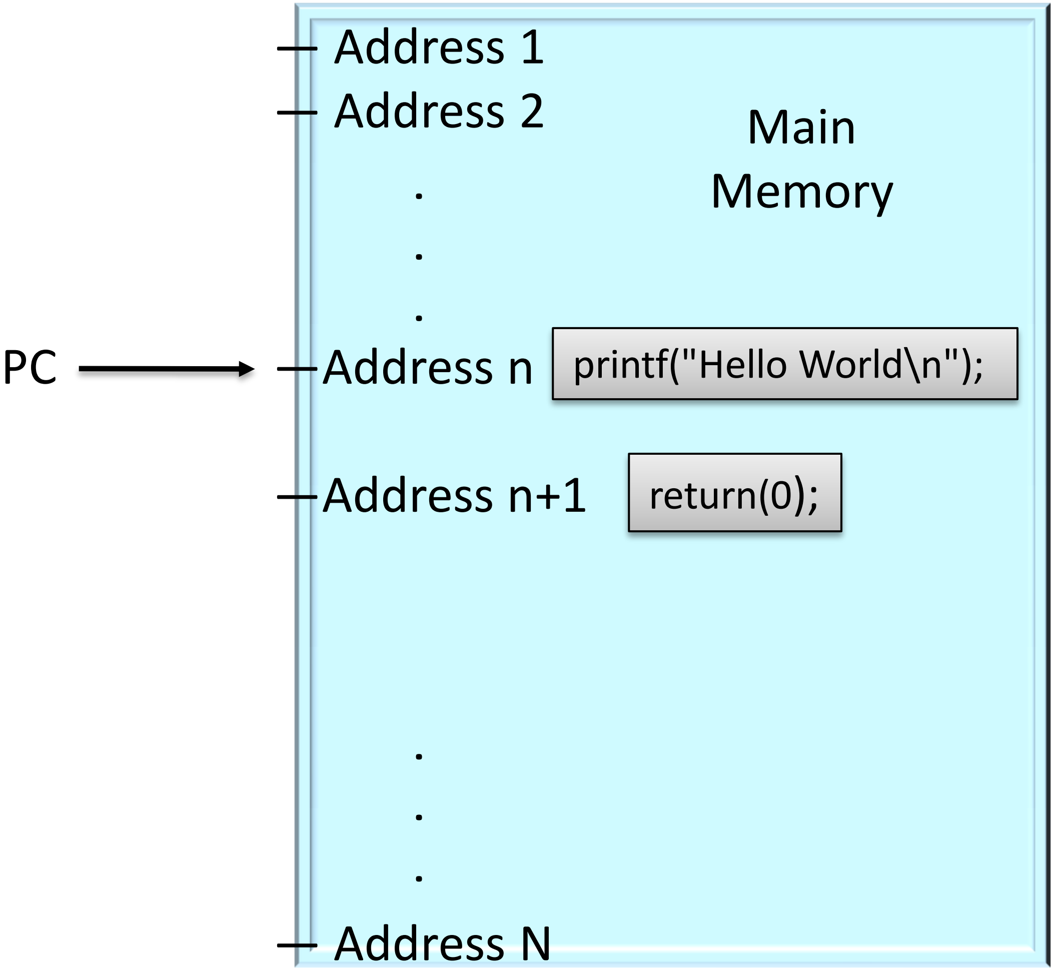
Hello World

```
#include <stdio.h>

int main(){

  printf("Hello World\n");

  return(0);
}
```



Von Neumann Architecture

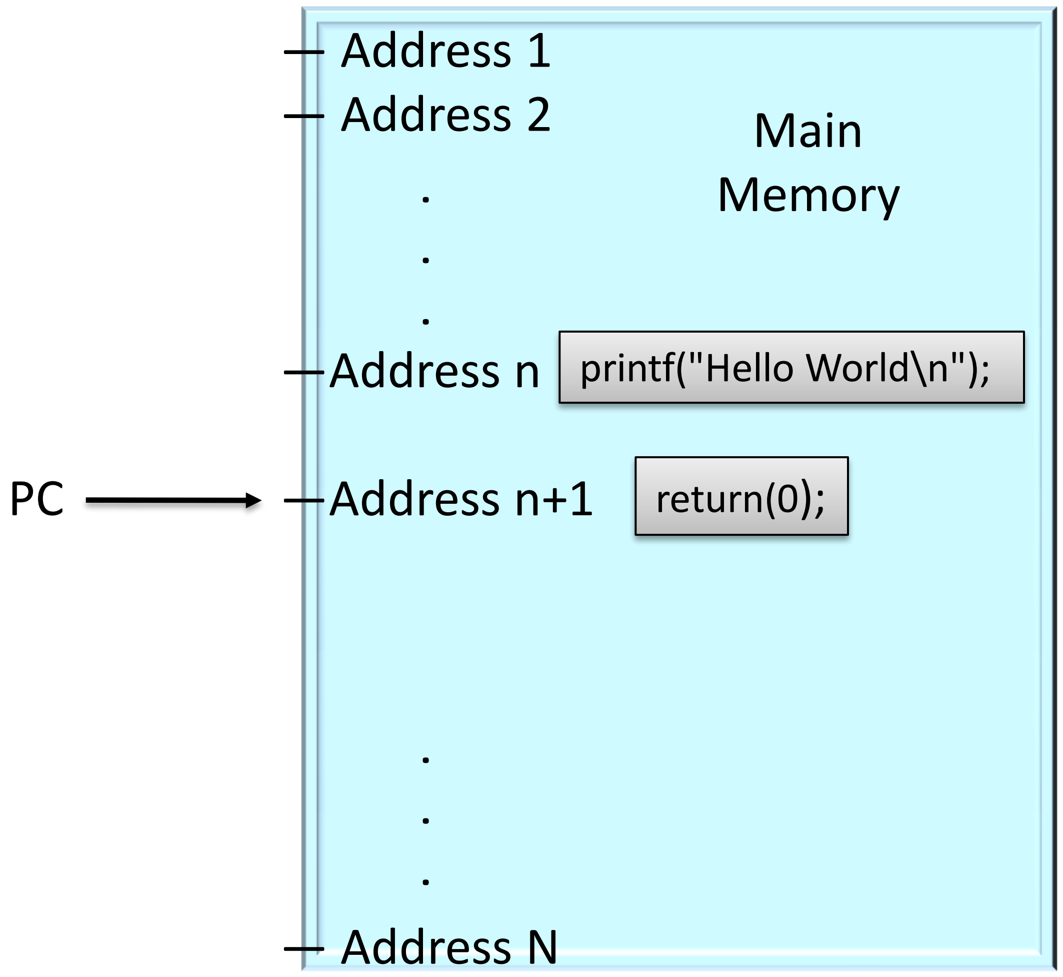
Hello World

```c
#include <stdio.h>

int main(){

  printf("Hello World\n");

  return(0);
}
```



PC

Address 1

Address 2

.

.

.

Address n printf("Hello World\n");

Address n+1 return(0);

.

.

.

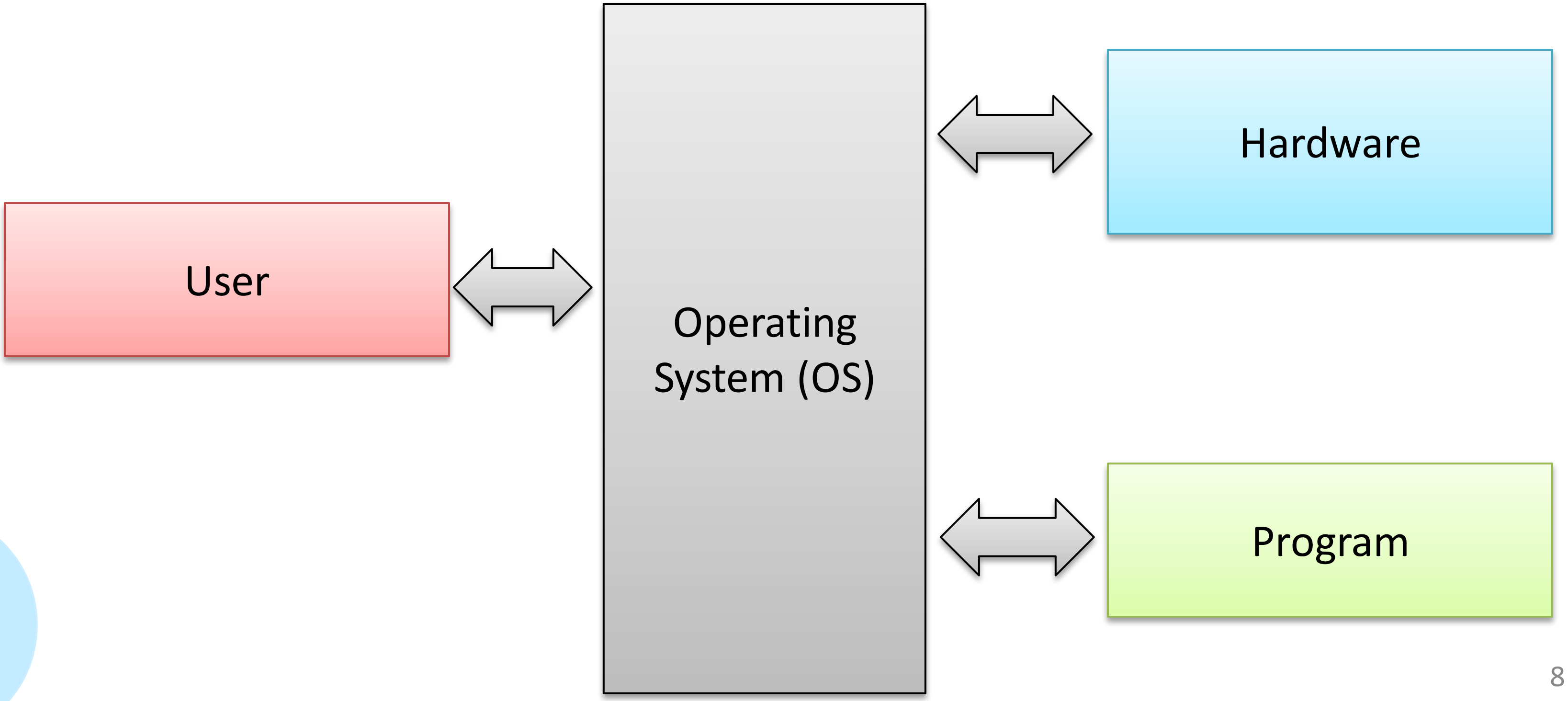
Address N

Main Memory

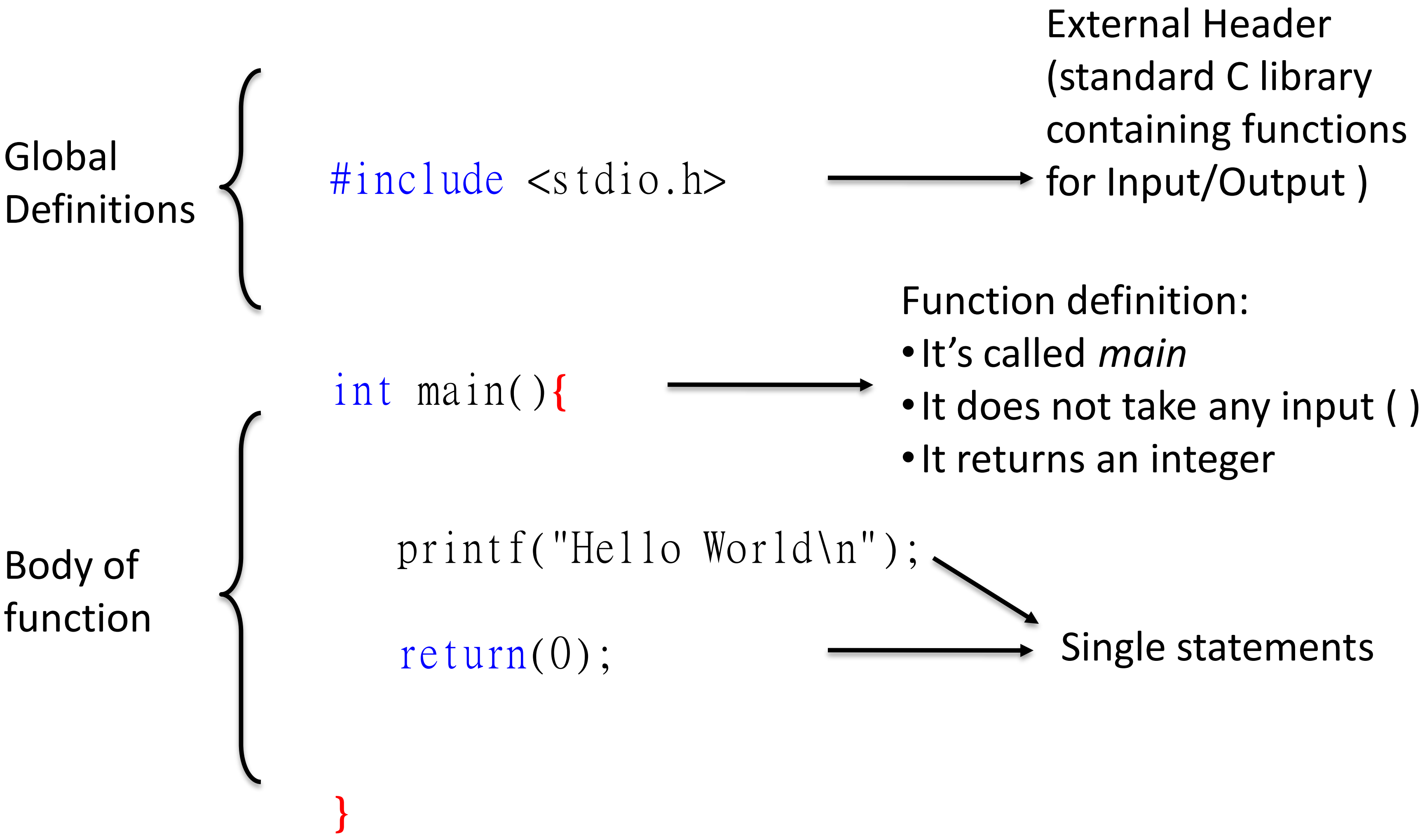
The Operating System

- Manages the hardware
- Allocates resources to programs
- Accommodates user requests
- First program to be executed when computer starts (loaded from ROM)

- Windows
- Unix
- Mac OS
- Android
- Linux
- Solaris
- Chrome OS



Hello World



Global Definitions

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

External Header (standard C library containing functions for Input/Output)

```c
int main(){
```

Function definition:

- It’s called *main*
- It does not take any input ( )
- It returns an integer

Body of function

```c
    printf("Hello World\n");

    return(0);

}
```

Single statements

C Syntax

- Statements
  - one line commands
  - always end with ;
  - can be grouped between { }
  - spaces are not considered

- Comments

  // single line comment

  /* multiple lines comments
  */

Hello World + Comments

```
/*
 *  My first C program
 */

#include <stdio.h>


int main(){


   printf(”Hello World\n”);

   return(0);   // return 0 to the OS = OK


}
```

Variables and types

- **Variables** are placeholders for values

```
int x = 2;
x = x + 3; // x value is 5 now
```

- In C, variables are divided into **types**, according to how they are **represented in memory** (always represented in binary)
  - **int**
  - **float**
  - **double**
  - **char**

Variables Declaration

- Before we can use a variable, we must **declare** (= create) it
- When we declare a variable, we specify its **type** and its **name**

```
int x;
float y = 3.2;
```

- Most of the time, the compiler also **allocates memory** for the variable when it’s declared. In that case **declaration = definition**
- There exist special cases in which a variable is declared but not defined, and the computer allocates memory for it only at run time (will see with functions and external variables)

int

- No fractional part or decimal point (ex. +3, -100)
- Represented with 4 bytes (32 bits) in UNIX
- <u>Sign</u>
  - **unsigned** : represents only positive values, all bites for value

    Range: from 0 to 2^32
  - **signed** (default) : 1 bit for sign + 31 for actual value

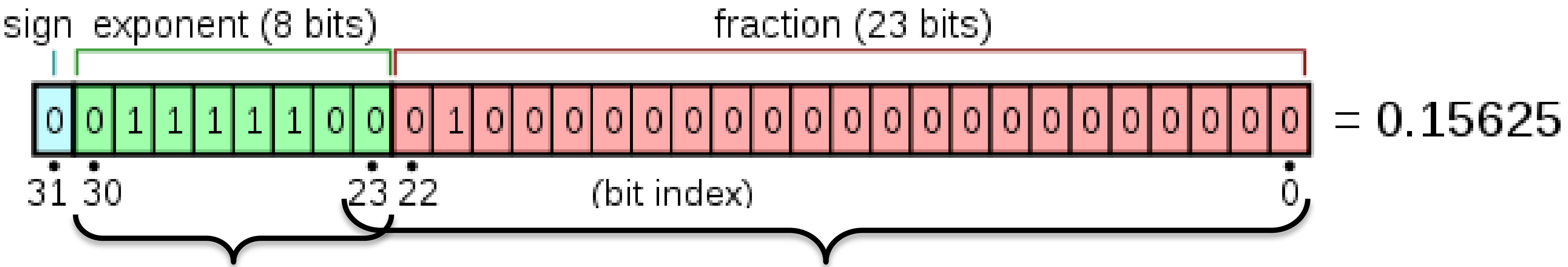
    Range: from -2^31 to 2^31
- <u>Size</u>
  - **short** int : at least 16 bits
  - **long** int : at least 32 bits
  - **long long** int : at least 64 bits
  - size(short) ≤ size(int) ≤ size(long)

```
int x = -12;
unsigned int x  = 5;
short (int) x = 2;
```

float

- Single precision floating point value
- Fractional numbers with decimal point
- Represented with 4 bytes (32 bits)
- Range: -10^(38) to 10^(38)
- Exponential notation : - $\underset{f}{0.278}$ * 10^$\underset{m}{3}$

```
float x = 11.5;
```



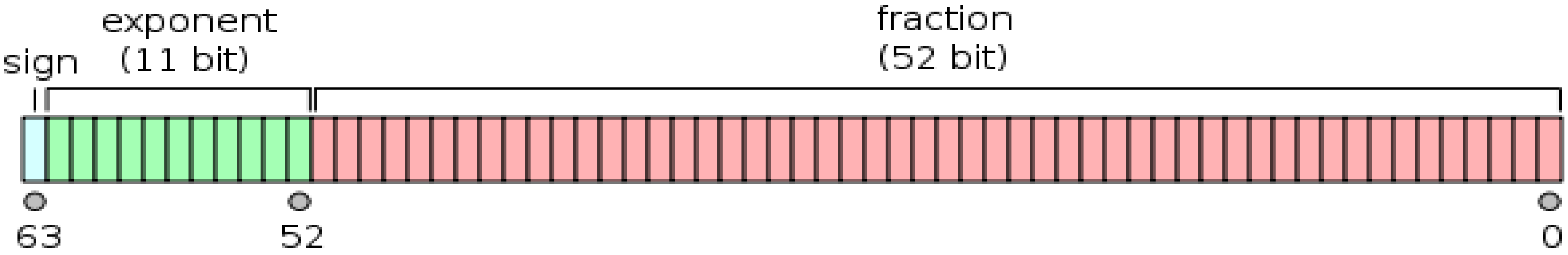
m

f: bit 23 is considered to be 1,
unless *m* is all zeros

$$n_{10} = (-1)^s \cdot (f \cdot 2^{-23}) \cdot 2^{m-127}$$

double

- Double precision floating point
- Represented with 8 bytes (64 bits)



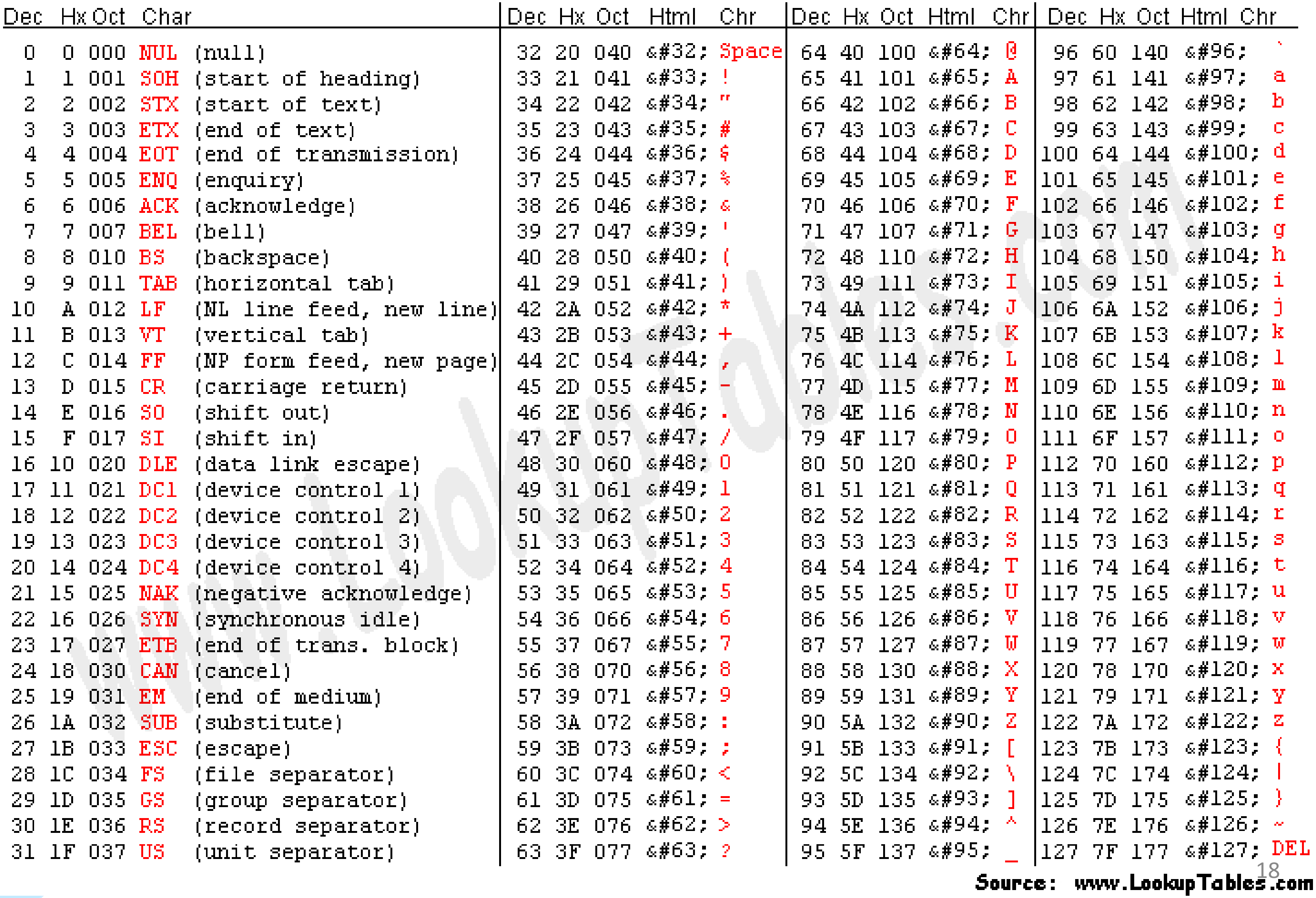
```
double x = 121.45;
```

char

- Character
- Single byte representation
- 0 to 255 values expressed in the ASCII table

```
char c = ‘w’ ;
```

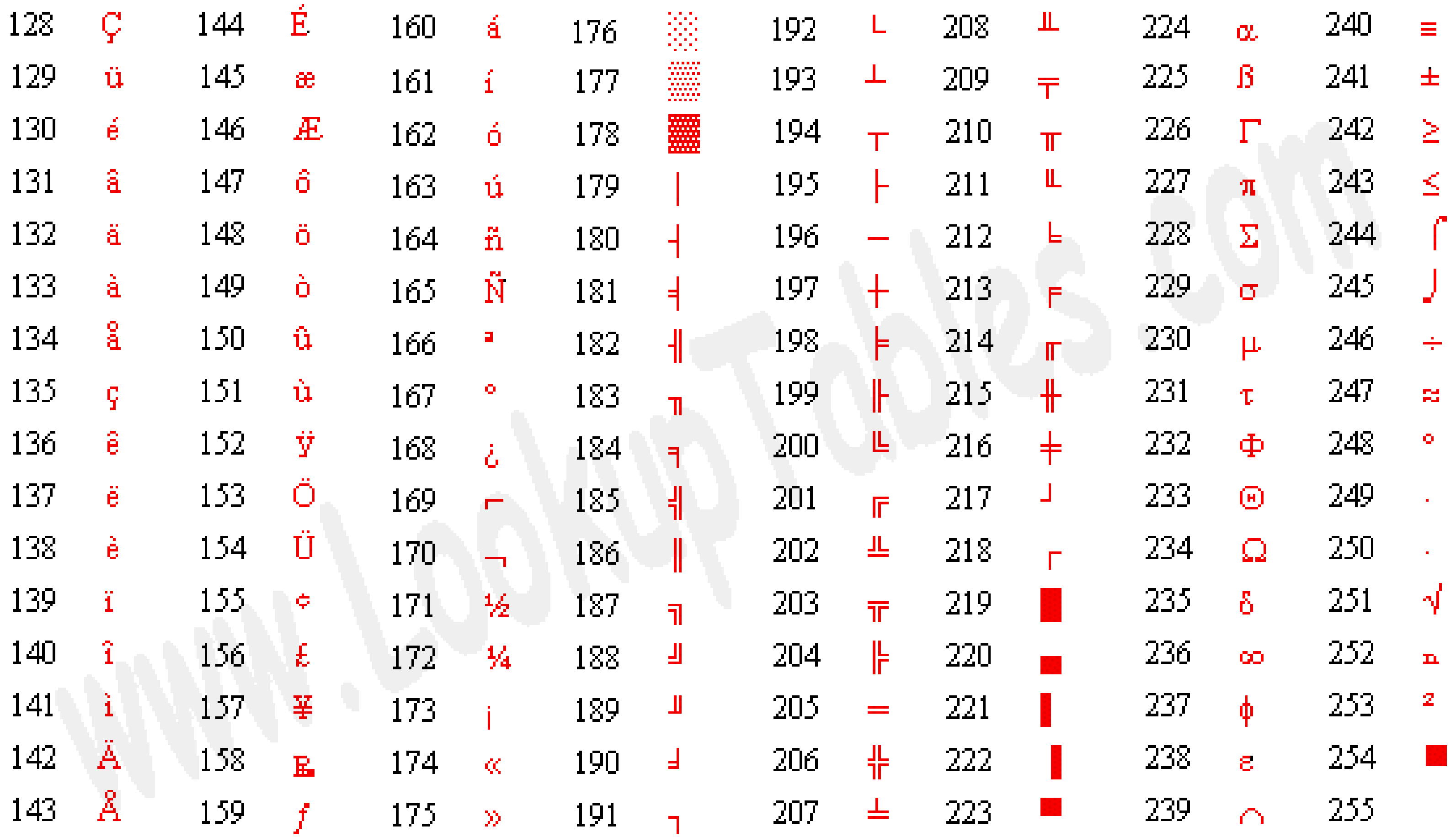
ASCII Table



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	C	C	99	63	143	&#99;	c
4	4	004	EOT (end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1 (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2 (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3 (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4 (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Extended ASCII Table



128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

Casting

- Casting is a method to correctly use variables of different types together
- It allows to treat a variable of one type as if it were of another type in a specific context
- When it makes sense, the compiler does it for us automatically

- Implicit (automatic)

```
int x = 1;
float y = 2.3;
x = x + y;
```

x= 3 compiler automatically casted (=converted) y to be an integer just for this instruction

- Explicit (non-automatic)

```
char c = ‘A’ ;
int x = (int) c;
```

Explicit casting from char to int. The value of x here is 65

Operators

- Assignment =
- Arithmetic * / % + -
- Increment ++ - - += -=
- Relational < <= > >= == !=
- Logical && | | !
- Bitwise & | ~ ^ << >>
- Comma ,

Operators – Assignment

```c
int x = 3;

x = 7;



int x, y = 5;

x = y = 7;

float y  = 2.3, z = 3, q = 700;



int i,j,k;

k = (i=2, j=3);

printf( “i = %d, j = %d, k = %d\n” ,i,j,k);
```

The comma operator allows us to perform multiple assignments/declarations

Operators - Arithmetic

* / % + -

- Arithmetic operators have a **precedence**

```
int x;

x = 3 + 5 * 2 - 4 / 2;
```

- We can use parentheses () to impose our precedence order

```
int x;

x = (3 + 5) * (2 – 4) / 2;
```

- % returns the module (or the remainder of the division)

```
int x;

x = 5 % 3;  // x = 2
```

- We have to be careful with integer vs. float division : remember automatic casting!

```
int x = 3;
float y;

y = x / 2; // y = 1.00
```

```
float y;
y = 1 / 2;  // y = 0.00
```

Operators - Arithmetic

`*   /   %   +   -`

- Arithmetic operators have a **precedence**

```
int x;

x = 3 + 5 * 2 - 4 / 2;
```

- We can use parentheses () to impose our precedence order

```
int x;

x = (3 + 5) * (2  –  4) / 2;
```

- % returns the module (or the remainder of the division)

```
int x;

x = 5 % 3;   // x = 2
```

- We have to be careful with integer vs. float division : remember automatic casting!

```
int x = 3;
float y;

y = x / 2; // y = 1.00
```

Possible fixes:
1)`float x  = 3;`
2)`y = (float) x /2;`
Then `y = 1.50`

```
float y;

y = 1 / 2;  // y = 0.00
```

Possible fix: `y = 1.0/2;`
Then `y = 0.50`

Operators - Increment

++   - -   +=   -=

```
int x = 3, y, z;
```

`x++;` → x is incremented at the end of statement

`++x;` → x is incremented at the beginning of statement

```
y = ++x + 3;  // x = x + 1; y = x + 3;

z = x++ + 3;  // z = x + 3; x = x + 1;

x -= 2;       // x = x - 2;
```

Operators - Relational

```
<   <=   >   >=   ==   !=
```

- Return 0 if statement is false, 1 if statement is true

```
int x = 3, y = 2, z, k, t;

z = x > y;        // z = 1

k = x <= y;       // k = 0

t = x != y;       // t = 1
```

Operators - Logical

&& || !

- A variable with value 0 is false, a variable with value !=0 is true

```
int x = 3, y = 0, z, k, t, q = -3;

z = x && y;     // z = 0;     x is true but y is false

k = x || y;     // k = 1;     x is true

t = !q;         // t = 0;     q is true
```

Review: Operators - Bitwise

- Work on the binary representation of data
- Remember: computers store and see data in binary format!

`int x, y, z , t, q, s, v;`

`x = 3;` 00000000000000000000000000000011

`y = 16;` 00000000000000000000000000010000

`z = x << 1;` equivalent to $z = x \cdot 2^{1}$ 00000000000000000000000000000110

`t = y >> 3;` equivalent to $t = y \cdot 2^{-3}$ 00000000000000000000000000000010

`q = x & y;` 00000000000000000000000000000000

`s = x | y;` 00000000000000000000000000010011

`v = x ^ y;` 00000000000000000000000000010011

↓ XOR

printf

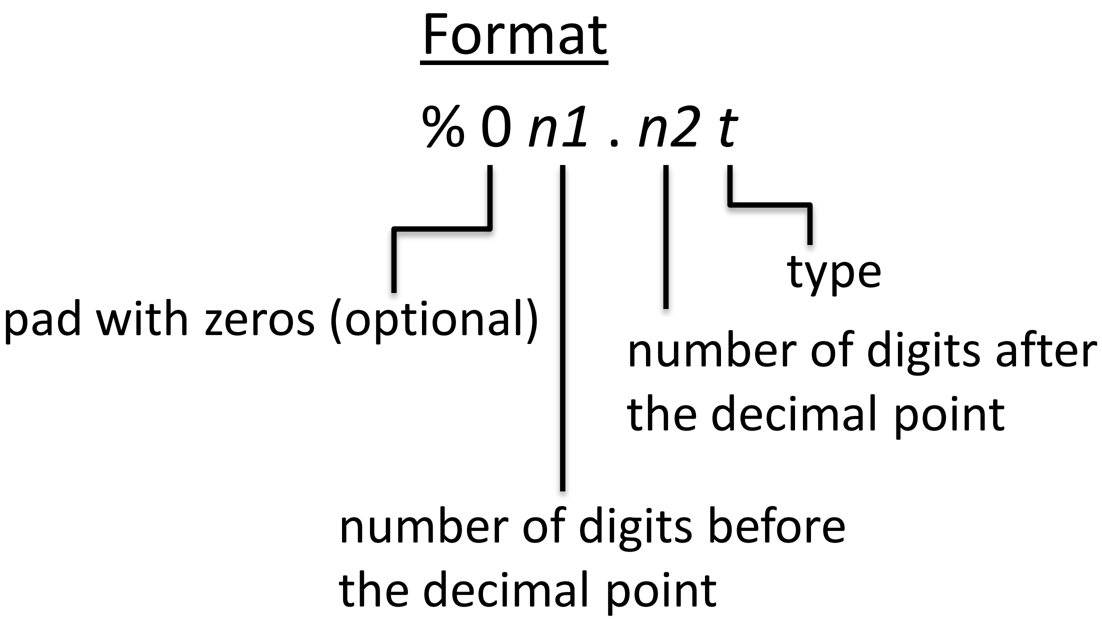
- `printf` is a function used to print to standard output (command line)

- Syntax:

```
printf(“format1 format2 …”, variable1, variable2,…);
```

- Format characters:

– %d or %i	integer
– %f	float
– %lf	double
– %c	char
– %u	unsigned
– %s	string



Format

% 0 *n1* . *n2* *t*

pad with zeros (optional)

number of digits before the decimal point

number of digits after the decimal point

type

printf

```
#include <stdio.h>

int main() {

   int a,b;
   float c,d;
   a = 15;
   b = a / 2;

   printf(”%d\n”,b);
   printf(”%3d\n”,b);
   printf(”%03d\n”,b);

   c = 15.3;
   d = c / 3;
   printf(”%3.2f\n”,d);

   return(0);

}
```

Output:

```
7
  7
007

5.10
```

printf

Escape sequences

\n	newline
\t	tab
\v	vertical tab
\f	new page
\b	backspace
\r	carriage return

Assignment

- Read PCP Chapter 3 and 4