Lecture 04

Introduction to C Programming

CSE115: Computing Concepts

General Form of a C Program

```
preprocessor directives
main function heading
{
   declarations
   executable statements
}
```

A Simple Program in C

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

int main()
{
    printf("Hello world!\n");
    return 0;
}
```

A Simple Program in C

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

stand**d**ard Library, **i**nput-**o**utput, **h**eader-file

Beginning of program

```
int main()
{
    printf("Hello world!\n");
    return 0;
}
```

Start of Segment

Function for printing text

End of statement

End of Segment

Preprocessor Directives

- A C program begins with # which provides an instruction to the C preprocessor
- It is executed **before** the actual compilation is done.
- Two most common directives :
  - `#include`
  - `#define`
- In our example (`#include<stdio.h>`) identifies the ***header*** file for standard input and output operations.

Function **main()**

- Identify the start of the program
- Every C program has a `main( )`
- 'main' is a C ***keyword.*** We **must not** use it for any other purpose.
- 4 common ways of main declaration

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

```
void main(void)
{

}
```

```
main(void)
{

}
```

```
main( )
{

}
```

The curly braces **{ }**

- Identify a ***segment / body*** of a program
  - The start and end of a function
  - The start and end of the selection or repetition block.
- Since the opening brace indicates the **start** of a segment with the closing brace indicating the **end** of a segment, **there must be just as many opening braces as closing braces** (this is a common mistake of beginners)

Statement

- Specifying an action to be taken by the computer as the program executes.
- Each statement in C <u>needs to be terminated with semicolon (;)</u>
- Example:

```
#include <stdio.h>
int main()
{
    printf("I love programming\n");
    printf("You will love it too once ");
    printf("you know the trick\n");
    return 0;
}
```

statement

statement

statement

statement

Statement

- Two types of statements:
  - Declaration
  - The part of the program that tells the compiler the names of memory cells in a program
  - Executable statements
  - Program lines that are converted to machine language instructions and executed by the computer

An Example

```
/*
Converts distance in miles
to kilometres.
*/
#include <stdio.h>   //printf, scanf definitions
#define KMS_PER_MILE 1.609 //conversion constant

int main(void) {
    float miles,   // input - distance in miles
          kms;     // output - distance in kilometres

    //Get the distance in miles
    printf("Enter distance in miles: ");
    scanf("%f", &miles);

    //Convert the distance to kilometres
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometres
    printf("That equals %f km.\n", kms);

    return 0;
}
```

An Example

```
/*
Converts distance in miles
to kilometres.
*/
#include <stdio.h>   //printf, scanf definitions
#define KMS_PER_MILE 1.609 //conversion constant

int main(void) {
    float miles,    // input - distance in miles
          kms;      // output - distance in kilometres

    //Get the distance in miles
    printf("Enter distance in miles: ");
    scanf("%f", &miles);

    //Convert the distance to kilometres
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometres
    printf("That equals %f km.\n", kms);

    return 0;
}
```

preprocessor directives

standard header file

reserved words

constant

variables

comments

functions

An Example

```
/*
Converts distance in miles
to kilometres.
*/
#include <stdio.h>   //printf, scanf definitions
#define KMS_PER_MILE 1.609 //conversion constant

int main(void) {
    declarations

    Executable statements

    return 0;
}
```

An Example

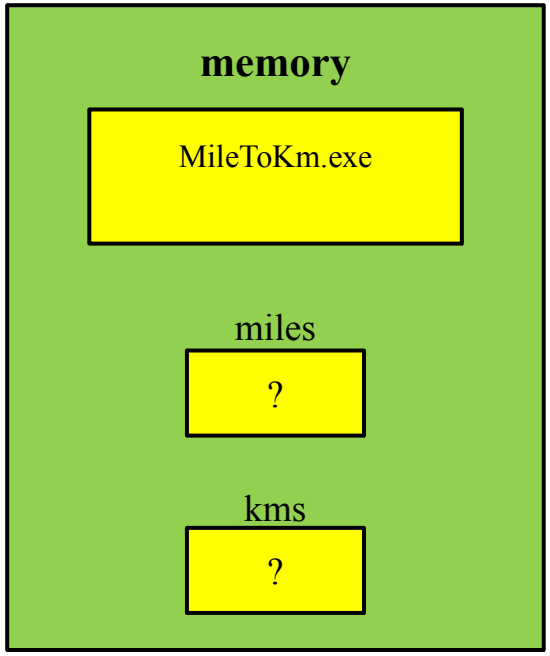
```
/*
Converts distance in miles
to kilometres.
*/
#include <stdio.h>   //printf, scanf definitions
#define KMS_PER_MILE 1.609 //conversion constant

int main(void) {
    float miles,   // input - distance in miles
          kms;     // output - distance in kilometres

    //Get the distance in miles
    printf("Enter distance in miles: ");
    scanf("%f", &miles);

    //Convert the distance to kilometres
    kms = KMS_PER_MILE * miles;

    //Display the distance in kilometres
    printf("That equals %f km.\n",

    return 0;
}
```

Sample Run

```
Enter distance in miles: 10.5
That equals     16.89 km.
```

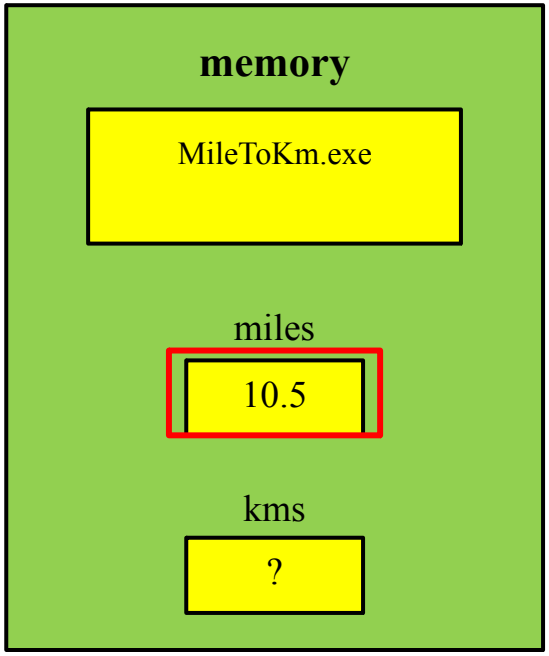
An Example

- What happens in the computer memory?



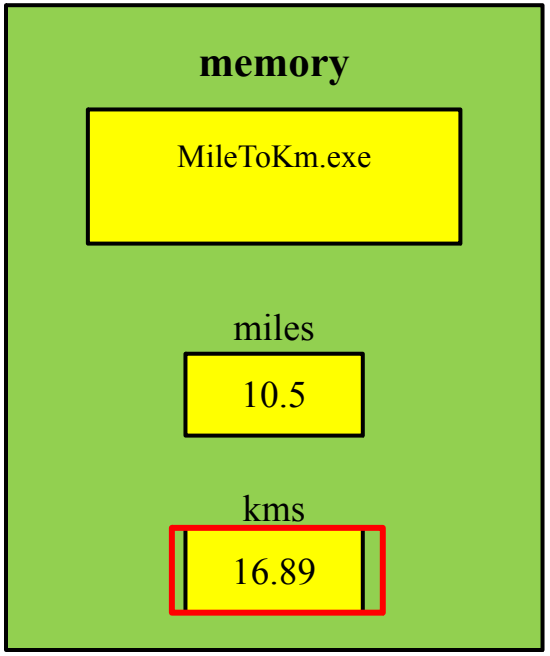
At the beginning

Do not assume that uninitialised variables contain zero! (Very common mistake.)



After user enters: 10.5 to

```
scanf("%f", &miles);
```



After this line is executed:

```
kms = KMS_PER_MILE * miles;
```

Variables

- **Variable** ▭ a name associated with a memory cell whose value can change
- **Variable Declaration**: specifies the type of a variable
  - Example: `int num;`
- **Variable Definition**: ***assigning*** a value to the declared ***variable***
  - Example: `num = 5;`

Basic Data Types

- There are 4 basic ***data types*** :
  - `int`
  - `float`
  - `double`
  - `char`

- **int**
  - used to declare numeric program variables of integer type
  - whole numbers, positive and negative
  - keyword: int
  - `int number;`
  - `number = 12;`

Basic Data Types

- **float**
  - fractional parts, positive and negative
  - keyword: float
  - `float height;`
  - `height = 1.72;`
  -
- **double**
  - used to declare floating point variable of higher precision or higher range of numbers
  - exponential numbers, positive and negative
  - keyword: double

```
double valuebig;
```

- `valuebig = 12E-3; (is equal to 12X10-3)`

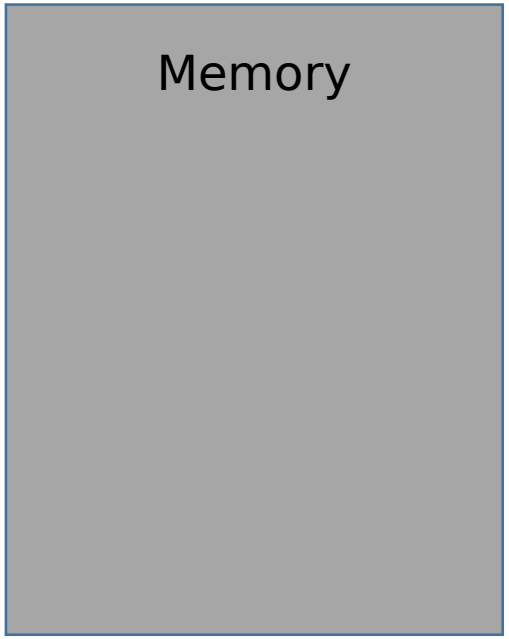
Basic Data Types

- **char**
  - equivalent to ‘letters’ in English language
  - Example of characters:
  - Numeric digits: `0 – 9`
  - Lowercase/uppercase letters: `a – z` and `A – Z`
  - Space (blank)
  - Special characters: `, . ; ? " / ( ) [ ] { } * & % ^ < >` etc
  - single character
  - keyword: char
  - `char my_letter;`
  - `my_letter = 'U';`

The declared character must be enclosed within a single quote!

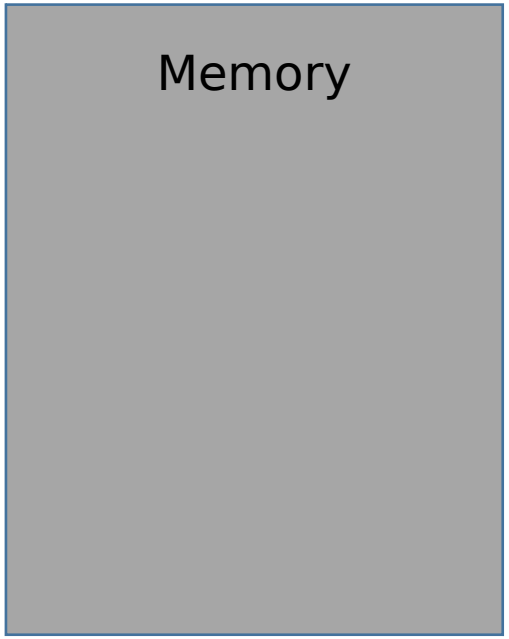
- In addition, there are **void, short, long,** etc.

A closer look at variables



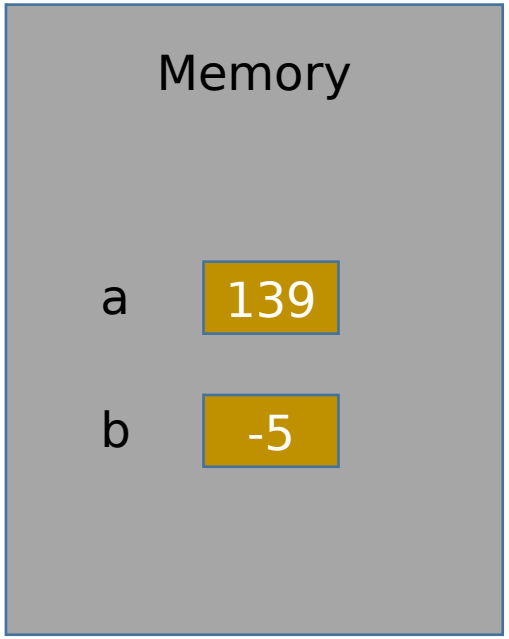
A closer look at variables

```
int a = 139, b = -5;
```



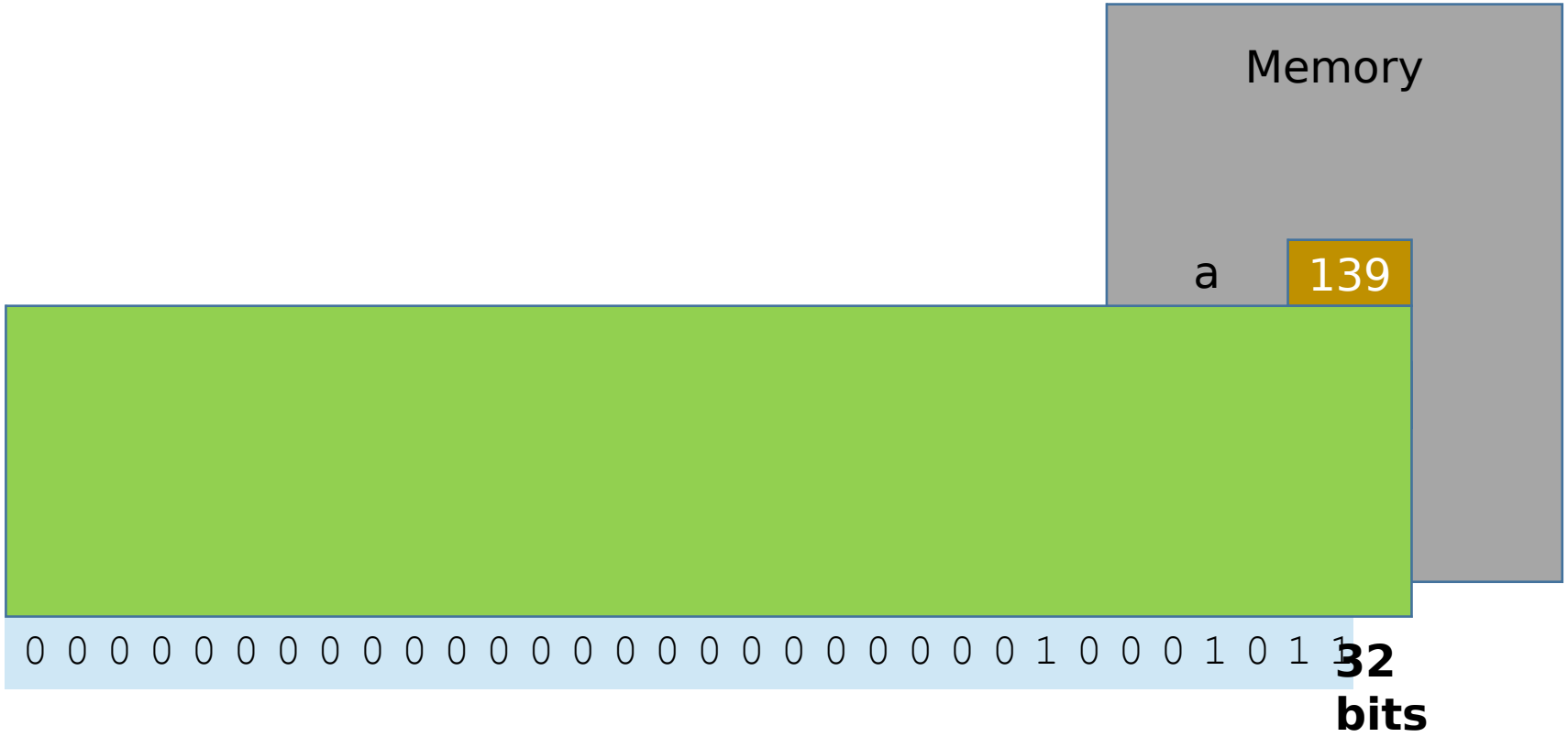
A closer look at variables

```
int a = 139, b = -5;
```



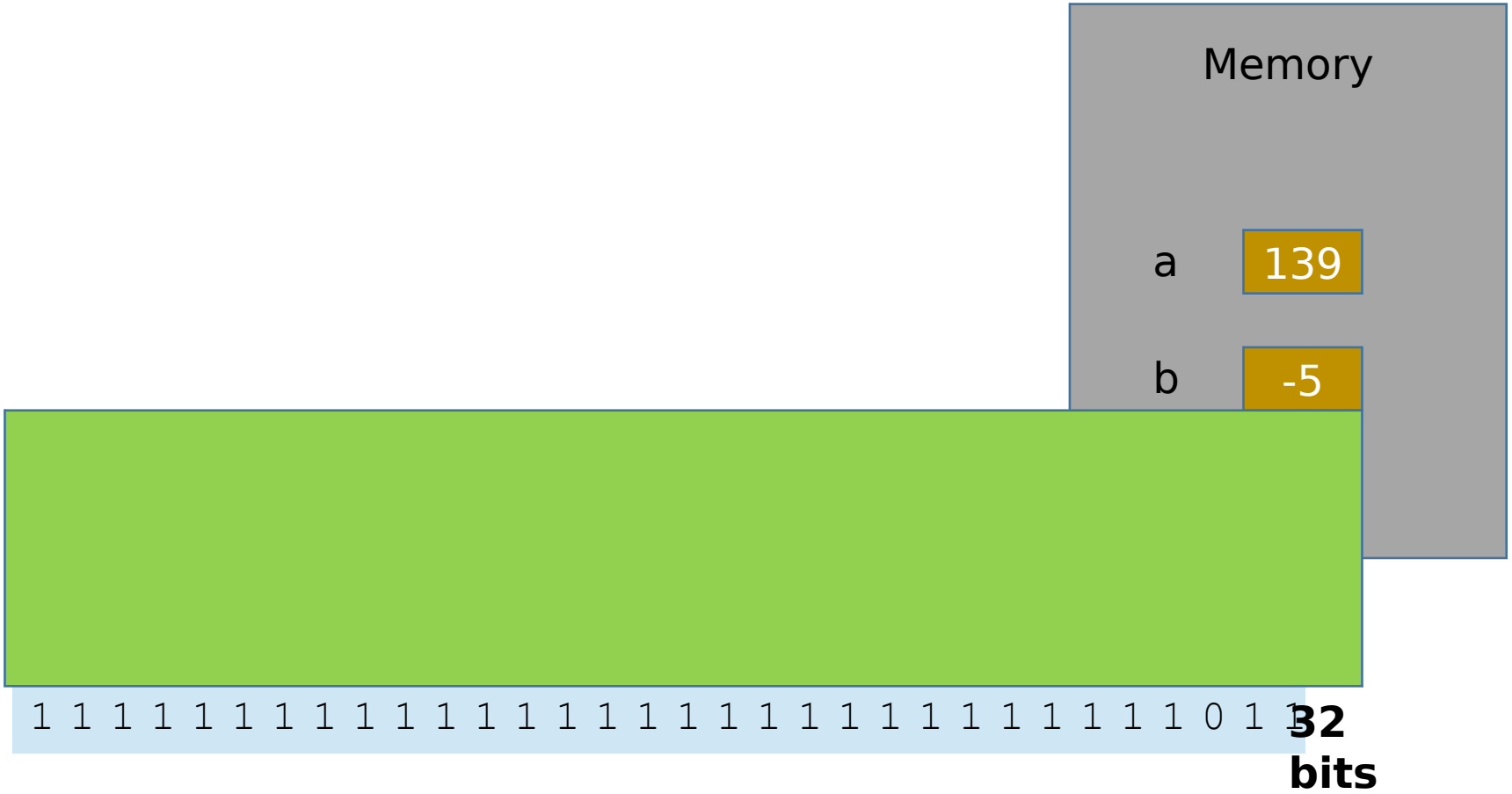
A closer look at variables

```
int a = 139, b = -5;
```



A closer look at variables

```
int a = 139, b = -5;
```



A closer look at variables

```
char c = 'H';
```

0	1	0	0	1	0	0	0

8 bits

A closer look at variables

```
char c = 'H';
```

0	1	0	0	1	0	0	0

8 bits

?

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□

A closer look at variables

```
char c = 'H';
```

0	1	0	0	1	0	0	0

8 bits

?

$10010002 = 7210$

Input/Output Operations

- Input operation
  - an instruction that copies data from an input device into memory
- Output operation
  - an instruction that displays information stored in memory to the output devices (such as the monitor screen)

Input/Output Functions

- A C function that performs an input or output operation
- A few functions that are pre-defined in the header file `stdio.h` such as :
  - `printf() - print something`
  - `scanf() - read something from user`
  - `getchar() - read a character from user`
  - `putchar() - print a character`

The `printf` function

- Used to send data to the standard output (usually the monitor) to be printed according to specific format.
- General format:
  - **printf(“string literal”);**
  - A sequence of any number of characters surrounded by double quotation marks.
  - **printf(“format string”, variables);**
  - Format string is a combination of text, conversion specifier and escape sequence.

The **`printf`** function

- Example:
  - `printf(“Thank you”);`
  - `printf (“Total sum is: %d\n”, sum);`
  - %d is a placeholder (conversion specifier)
  - marks the display position for an integer type variable
  - \n is an escape sequence
  - moves the cursor to the new line

Placeholder / Conversion Specifier

No	Conversion Specifier	Output Type	Output Example
1	%d	Signed decimal integer	76
2	%i	Signed decimal integer	76
3	%o	Unsigned octal integer	134
4	%u	Unsigned decimal integer	76
5	%x	Unsigned hexadecimal (small letter)	9c
6	%X	Unsigned hexadecimal (capital letter)	9C
7	%f	Integer including decimal point	76.0000
8	%e	Signed floating point (using e notation)	7.6000e+01
9	%E	Signed floating point (using E notation)	7.6000E+01
10	%g	The shorter between %f and %e	76
11	%G	The shorter between %f and %E	76
12	%c	Character	‘7’
13	%s	String	‘76'

Escape Sequence

Escape Sequence	Effect
\a	Beep sound
\b	Backspace
\f	Formfeed (for printing)
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\\	Backslash
\”	“ sign
\o	Octal decimal
\x	Hexadecimal
\O	NULL

Formatting output

```
int meters = 21, feet = 68 , inches =
11;
printf("Results: %3d meters = %4d ft.
%2d in.\n", meters, feet, inches);
```

```
Results:  21 meters =   68 ft. 11 in.
```

```
printf("Results: %03d meters = %04d ft.
%02d in.\n", meters, feet, inches);
```

```
Results: 021 meters = 0068 ft. 11 in.
```

Formatting output

Value	Format	Displayed Output
234	%4d	▮234
234	%5d	▮▮234
234	%6d	▮▮▮234
234	%1d	234

Value	Format	Displayed Output
-234	%4d	-234
-234	%5d	▮-234
-234	%6d	▮▮-234
-234	%2d	-234

Formatting output

- **Displaying x Using Format String Placeholder %6.2f**

Value of x	Displayed Output	Value of x	Displayed Output
-99.42	-99.42	-25.554	-25.55
.123	▮▮0.12	99.999	100.00
-9.536	▮-9.54	999.4	999.40

The `scanf` function

- Read data from the standard input device (usually keyboard) and store it in a variable.
- General format:
  - **scanf("Format string", &variable);**
- Notice ampersand (&) operator :
  - C address of operator
  - it passes the address of the variable instead of the variable itself
  - tells the scanf() where to find the variable to store the new value

The **scanf** function

- Example :

```
int age;
printf("Enter your age: ");
scanf("%d", &age);
```

- Common Conversion Identifier used in printf and scanf functions.

	printf	scanf
int	%d	%d
float	%f	%f
double	%lf	%lf
char	%c	%c
string	%s	%s

The **scanf** function

- If you want the user to enter more than one value, you serialize the inputs.
- Example:

```
float height, weight;

printf("Please enter your height and weight:");
scanf("%f%f", &height, &weight);
```