C Developer

Discover the C Syntax

SUPINFO

Course Objectives

- ✓ Declare and handle variables
- ✓ Manage inputs and outputs

Course Plan

1. Code Structure
2. Variables
3. Preprocessor Directives
4. Operators

1. Code Structure

1. Code Structure

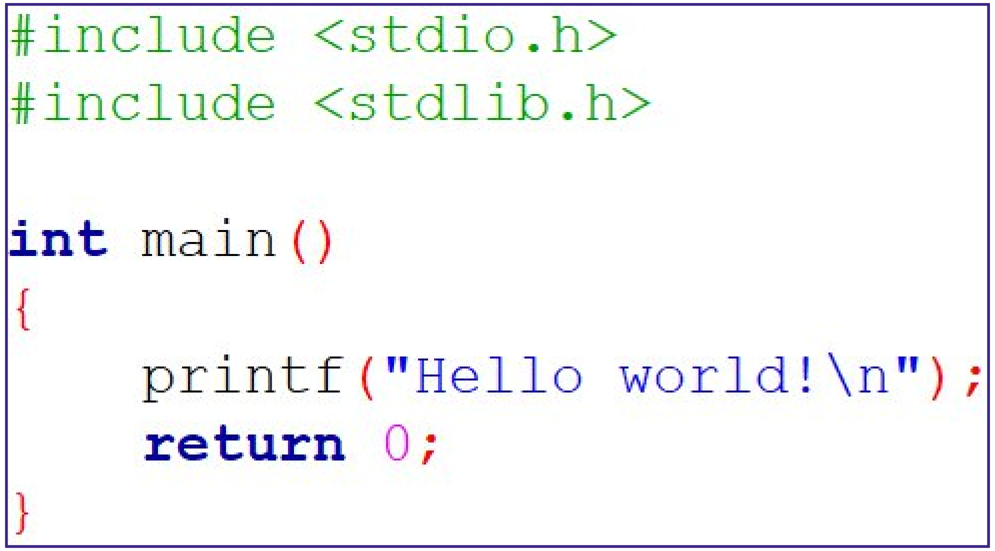
Main

- It is the entry point of the C program
- It must be unique
- Each instruction ends with “;”

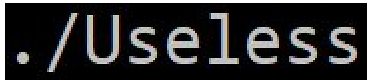
1. Code Structure

Main

Start the program without argument



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

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

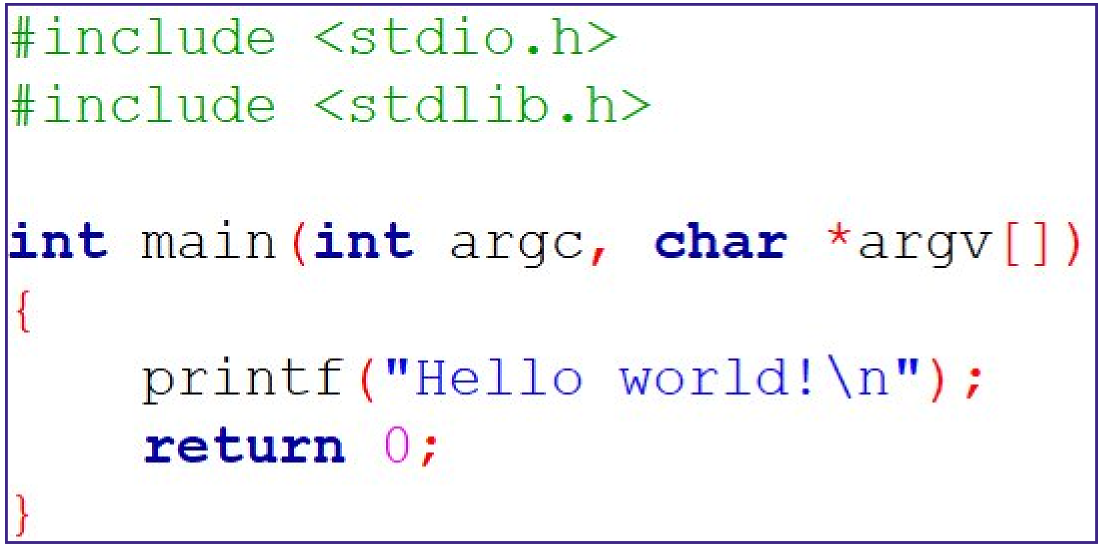


```
./Useless
```

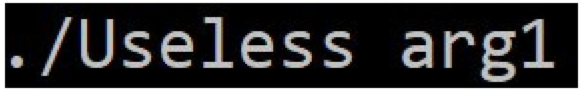
1. Code Structure

Main

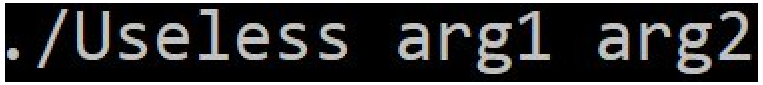
Start the program with argument(s)



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

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



```
./Useless arg1
```



```
./Useless arg1 arg2
```

1. Code Structure

Include Libraries

- A program contains many instructions
  - **#include** preprocessor directive
  - **<file.h>** compiler include

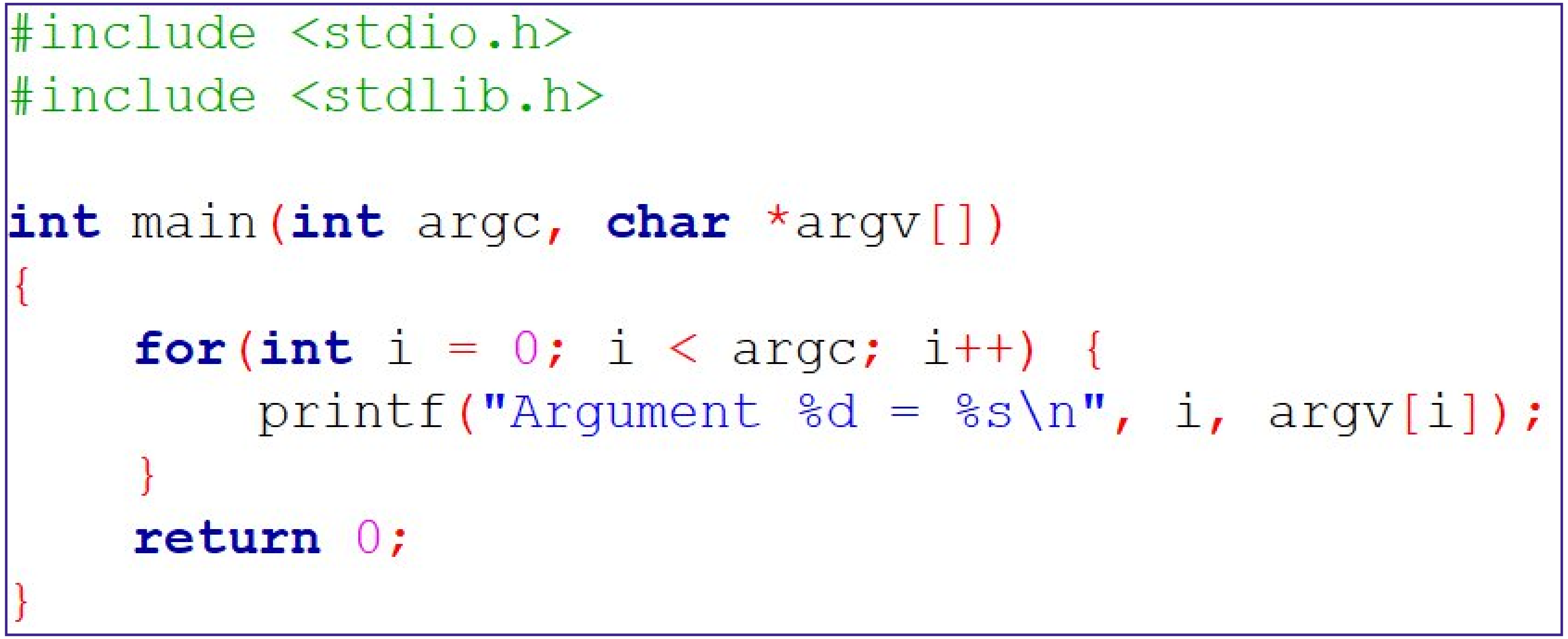
- **<stdio.h>**
  - Stands for Standard Input Output
  - Has the information related to input/output functions

- **<stdlib.h>**
  - Stands for Standard Library
  - Has the information of memory allocation/freeing functions

1. Code Structure

Include Libraries

With the default header files, we can print arguments

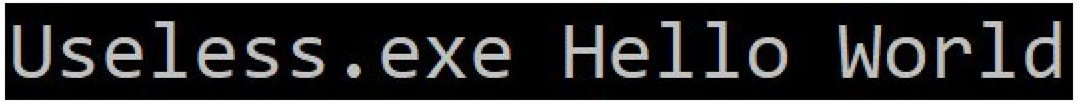


```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    for(int i = 0; i < argc; i++) {
        printf("Argument %d = %s\n", i, argv[i]);
    }
    return 0;
}
```

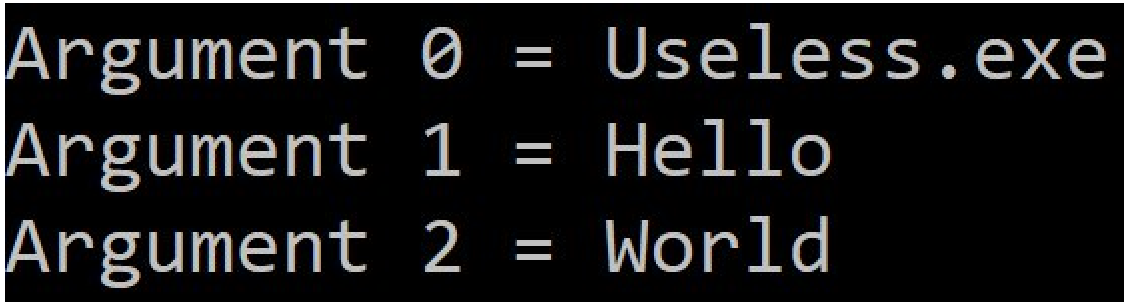
1. Code Structure

Include Libraries

With the default header files, we can print arguments



```
Useless.exe Hello World
```

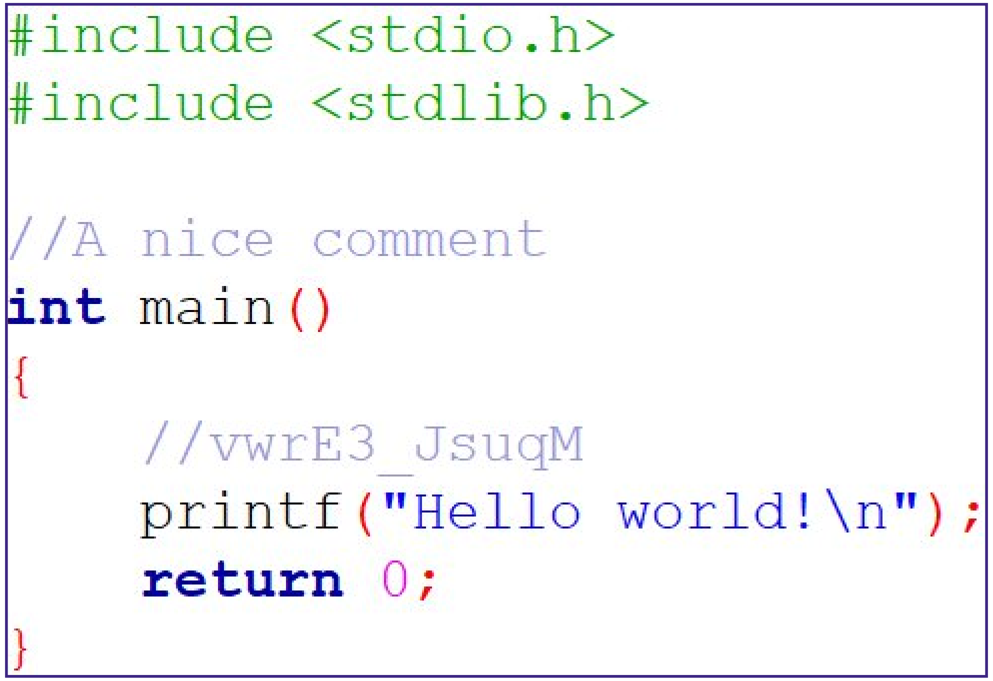


```
Argument 0 = Useless.exe
Argument 1 = Hello
Argument 2 = World
```

1. Code Structure

Comments

To insert a comment, use "**//**"

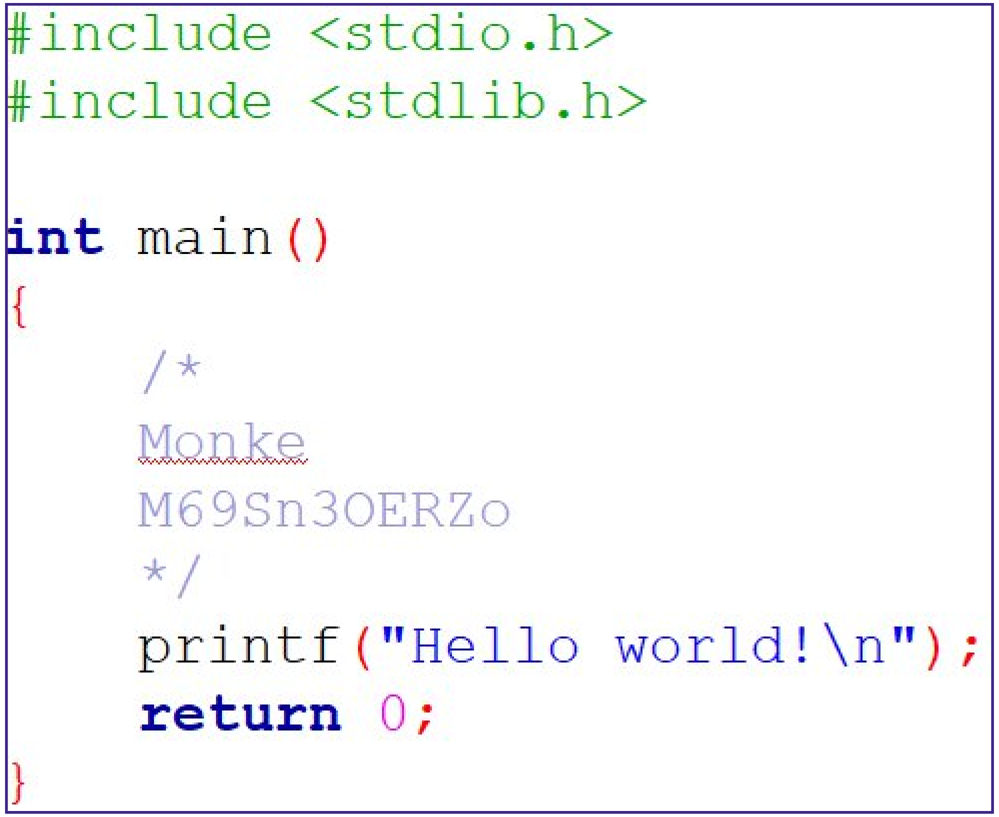


```
#include <stdio.h>
#include <stdlib.h>

//A nice comment
int main()
{
    //vwrE3_JsuqM
    printf("Hello world!\n");
    return 0;
}
```

1. Code Structure

Comments

To insert a block of comments, use “**/* … */**”



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

int main()
{
    /*
    Monke
    M69Sn3OERZo
    */
    printf("Hello world!\n");
    return 0;
}
```

1. Code Structure

Exercise

- Create a new Code::Blocks C project named “*HelloArgument*”
- Print “*Hello SUPINFO!*”
- Print only 5 arguments

1. Code Structure

Questions

2. Variables

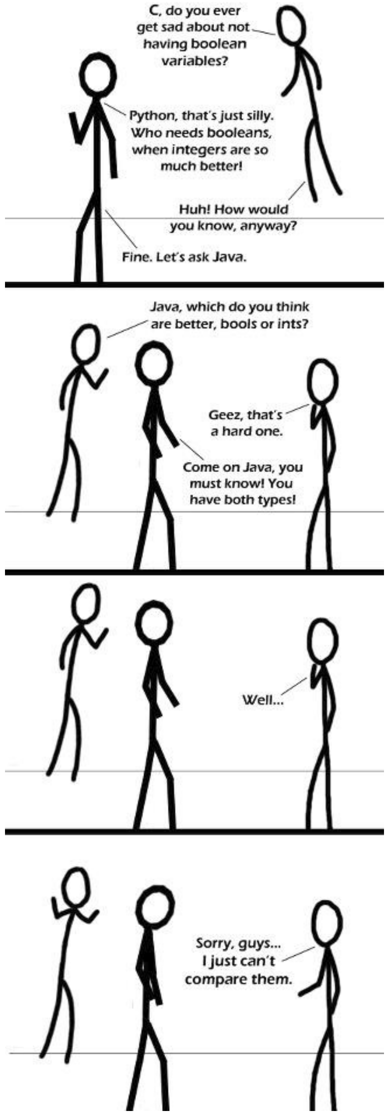
2. Variables

Introduction

To use information, you must know how it is encoded in memory:

- Integer
- Decimal
- Character

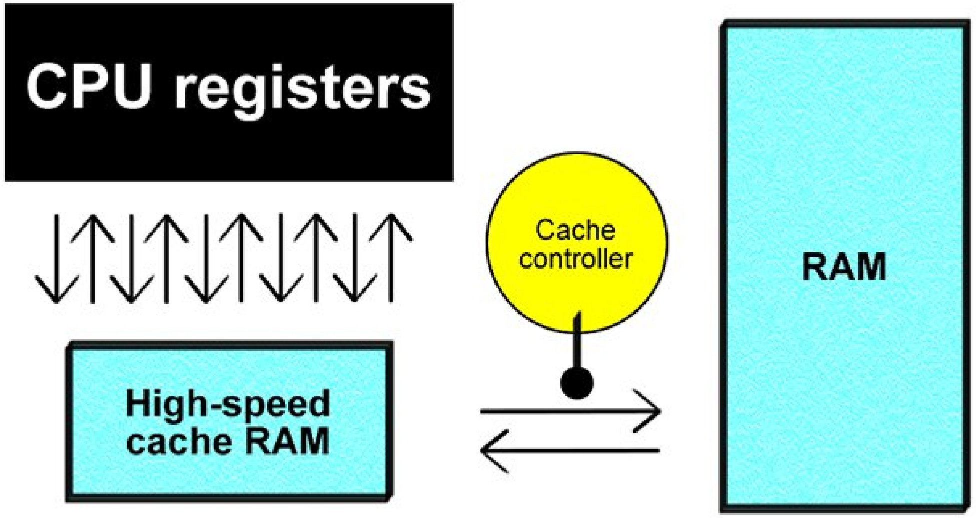
There is no boolean type!


C, do you ever get sad about not having boolean variables?
Python, that's just silly. Who needs booleans, when integers are so much better!
Huh! How would you know, anyway?
Fine. Let's ask Java.
Java, which do you think are better, bools or ints?
Geez, that's a hard one.
Come on Java, you must know! You have both types!
Well...
Sorry, guys... I just can't compare them.

2. Variables

Introduction

- All variables are numbers in memory
- Memories are binary (Register, Cache, RAM)
- Information is stored by a sequence of 0 and 1



2. Variables

Introduction

A representation of the memory

Address	Value
0000	1337
0001	
0002	a
0003	8
0004	
0005	
0006	
0007	42

2. Variables

Integer

Type	Size (min)	Min	Max
short	2 bytes/16 bits	-32 768	+32 767
unsigned short	2 bytes/16 bits	0	+65 535
int	4 bytes/32 bits*	-2 147 483 648	+2 147 483 647
unsigned int	4 bytes/32 bits*	0	+4 294 967 295
long	4 bytes/32 bits	-2 147 483 648	+2 147 483 647
unsigned long	4 bytes/32 bits	0	+4 294 967 295
long long (C99)	8 bytes/64 bits	-9 223 372 036 854 775 808	+9 223 372 036 854 775 807
unsigned long long (C99)	8 bytes/64 bits	0	+18 446 744 073 709 551 615

2. Variables

Integer

Example of **unsigned short**:

Decimal	Binary (in memory)	Hexadecimal
1	00000000 00000001	0001
2	00000000 00000010	0002
3	00000000 00000011	0003
…	…	…
65 534	11111111 11111110	FFFE
65 535	11111111 11111111	FFFF

2. Variables

Integer

Example of **signed short**:

Decimal	Binary (in memory)	Hexadecimal
32 767	01111111 11111111	7FFF
…	…	…
1024	00000100 00000000	0400
…	…	…
0	00000000 00000000	0000
-1	11111111 11111111	FFFF
…	…	…
-1024	11111100 00000000	FC00
…	…	…
-32 768	10000000 00000000	8000

2. Variables

Integer

- A good developer does not allocate more memory than necessary
- The program will be more efficient
- Choose the right type of variable

2. Variables

Decimal

Type	Size (min)	Precision (decimal)	Min	Max
float	4 bytes/32 bits	6	1.2E-38	3.4E+38
double	8 bytes/64 bits	15	2.3E-308	1.7E+308
long double	12 bytes/96 bits*	18	3.4E-4932	1.1E+4932

Real numbers are represented either in decimal form or using a power of 10:

280.61 = 2.8061e2 = 28061E-2

2. Variables

Character

- A character is a number
  - The letter “**A**” (uppercase) is stored as **65** in the memory (ASCII)

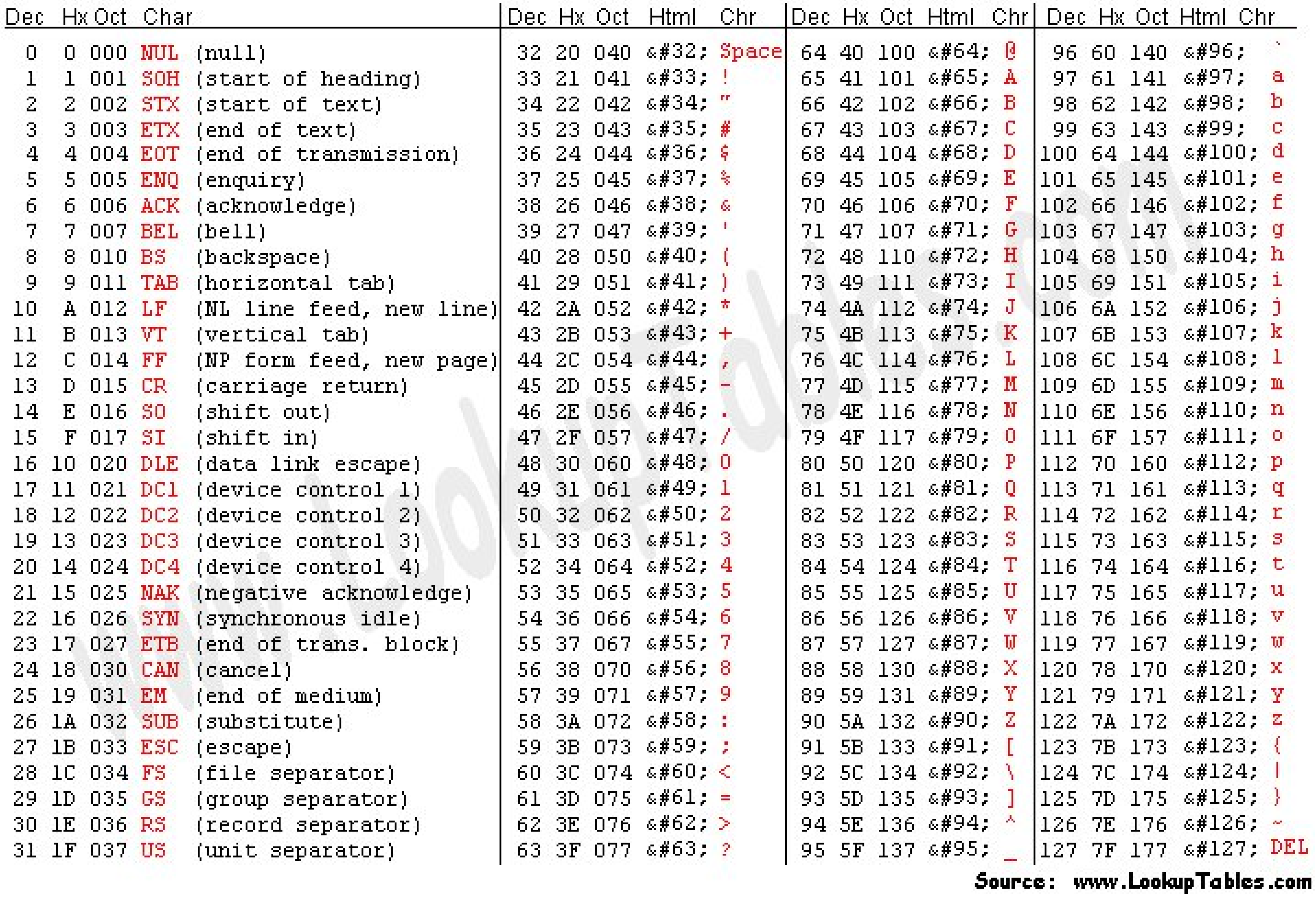
- A **char** is a representation of a character of an integer from **-128** to **127**

- An **unsigned char** is a representation of a character of an integer from **0** to **255**

2. Variables

Character

ASCII table



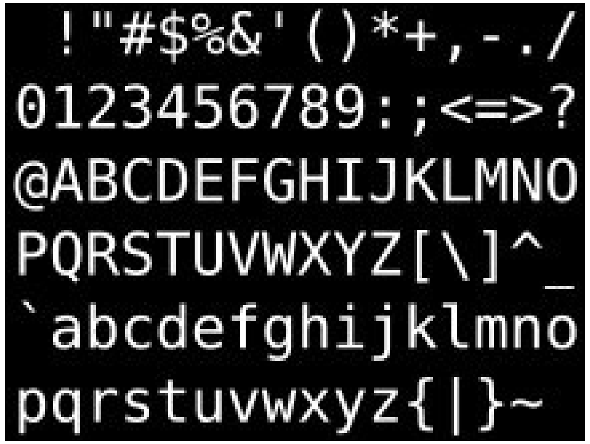
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	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

2. Variables

Character

- The characters from 32 to 126 can be displayed as follows:



```
 !"#$%&'()*+,-./
0123456789:;<=>?
@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_
`abcdefghijklmno
pqrstuvwxyz{|}~
```

- To manipulate a character, you can use “ **' … '** ”
  - **36** and **'$'** are identical (just like **'\x24'** for hexadecimal lovers)

- String handling is a bit tricky

2. Variables

Character

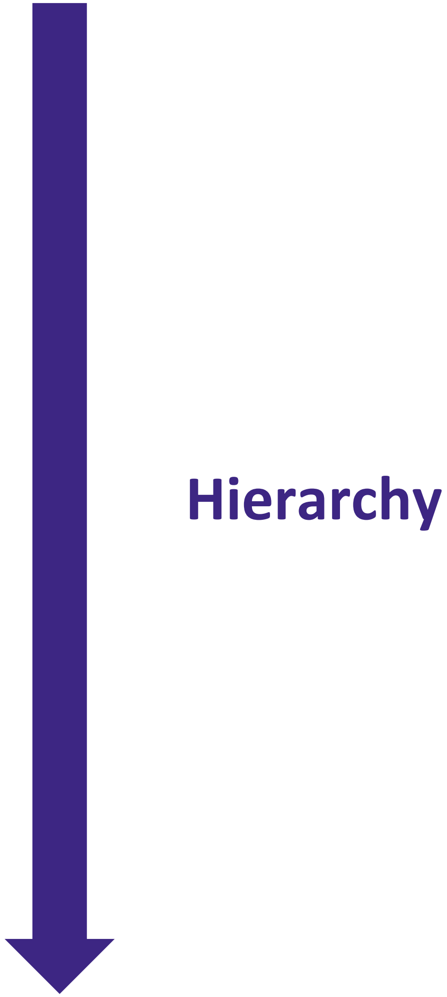
Some non-displayable characters can also be designated without their ASCII code, using a “\” between quotes

Escape Sequence	ASCII Code (Decimal)	ASCII Code (Hexadecimal)	Meaning
\n	10	A	New Line
\t	9	9	Horizontal Tab
\b	8	8	Backspace
\r	13	D	Carriage Return
\a	7	7	(Alert) Bell
\'	39	27	Single Quote
\"	34	22	Double Quote
\?	63	3F	Question Mark
\\\\	92	5C	Backslash
\f	12	C	Form Feed
\v	11	B	Vertical Tab
\0	0	0	Null Character

2. Variables

Summary

- char
- unsigned char
- short
- unsigned short
- int
- unsigned int
- long
- unsigned long
- long long
- unsigned long long
- float
- double
- long double



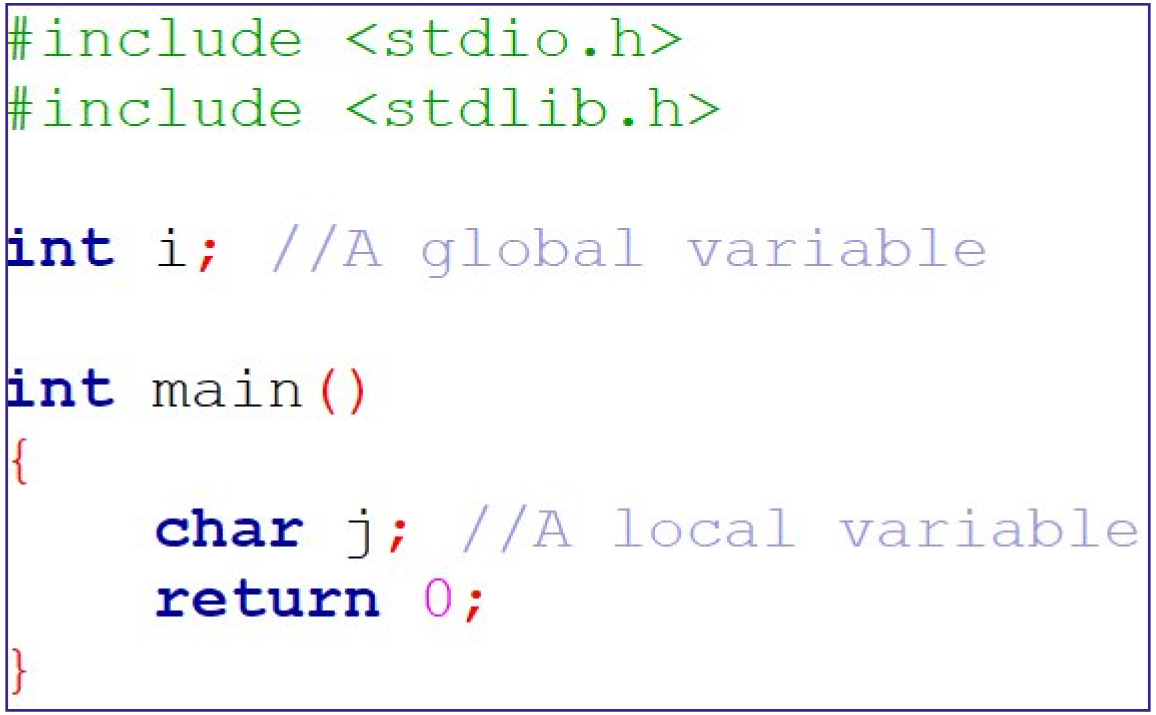
Hierarchy

2. Variables

Declaration and assignment

Variable can be declared as:

- Global (available anywhere)
- Local (available in the current scope)



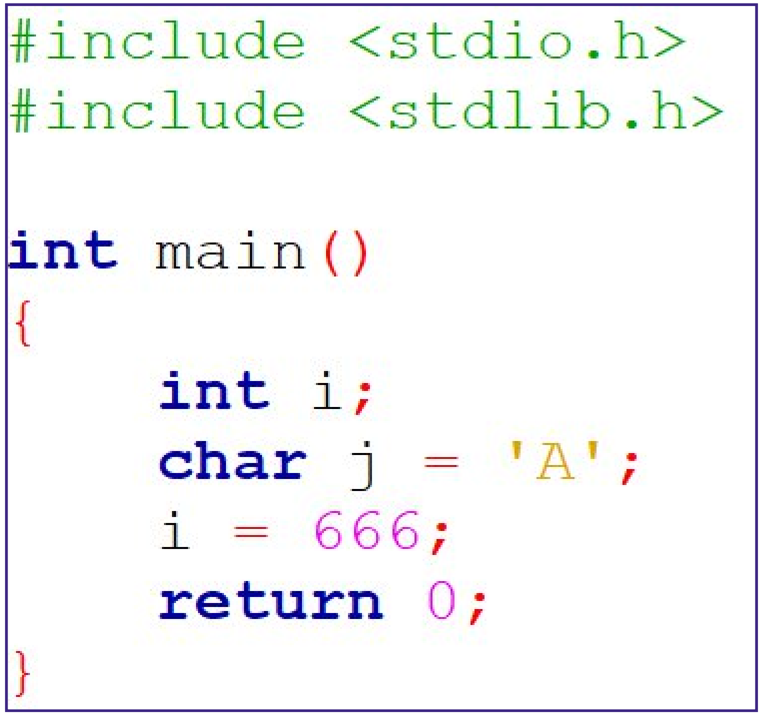
```
#include <stdio.h>
#include <stdlib.h>

int i; //A global variable

int main()
{
    char j; //A local variable
    return 0;
}
```

2. Variables

Declaration and assignment

The assignment:

- can be made during the declaration or later in the program
- is done with the “**=**” operator

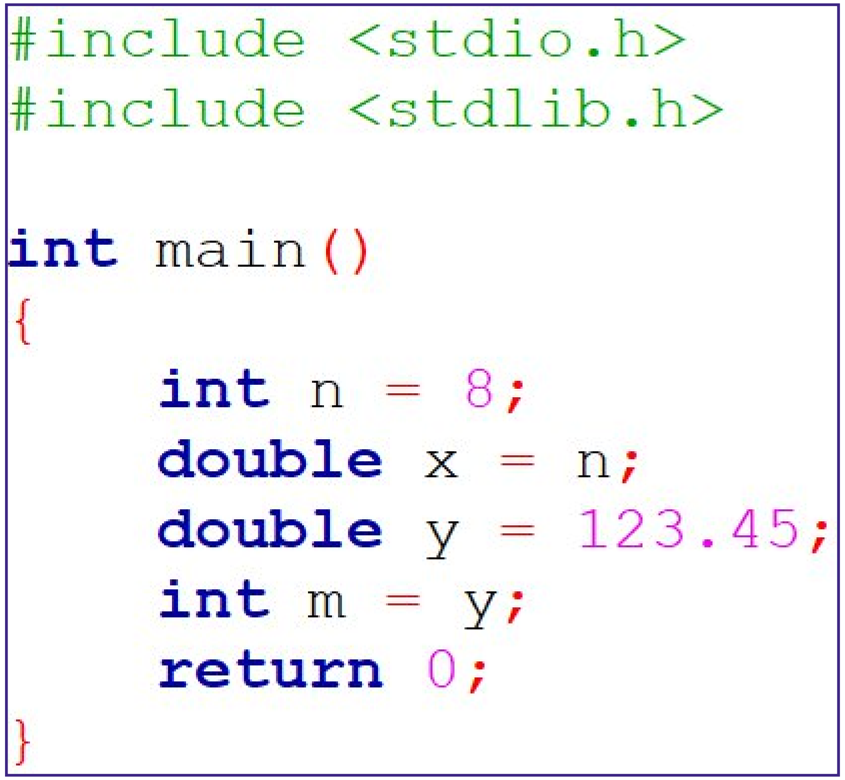


```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    char j = 'A';
    i = 666;
    return 0;
}
```

2. Variables

Declaration and assignment

- You can assign the value of a variable of a certain type to a variable of another type
- This implicit conversion is carried out without or with loss of information



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

int main()
{
    int n = 8;
    double x = n;
    double y = 123.45;
    int m = y;
    return 0;
}
```

n = 8
x = 8
y = 123.45
m = 123

2. Variables

Convention

- The name of a variable is referred as an identifier

- It must follow some rules:
  - It can contain letters (uppercase or lowercase), digits and underscores "**_**"
  - The first character must be a letter or an underscore (but avoid it for the last)
  - Most C compilers have a limit of 255 characters for an identifier, but ANSI standard recognizes a length of **31 characters** for a variable name

- Underscore is used to separate elements of an identifier

- Uppercase letters are recommended for ease of reading

2. Variables

Convention

- Do not use reserved keywords!

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

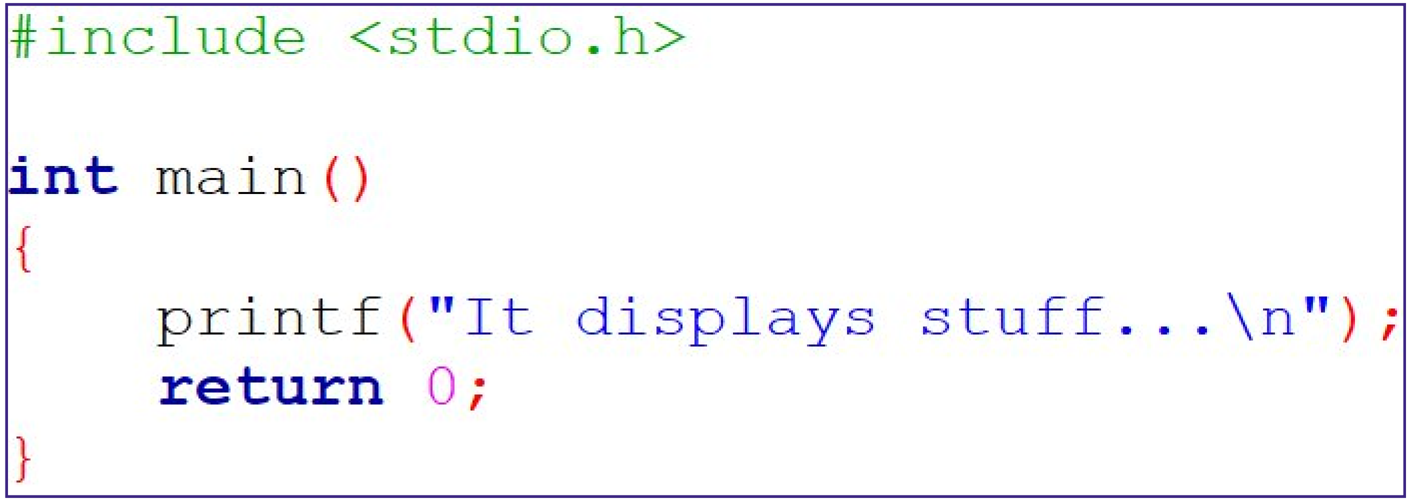
- C variable is case-sensitive

- Use a descriptive variable name and prefer to start with a lowercase letter

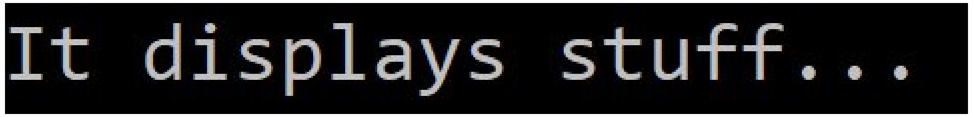
2. Variables

Display a variable

- Use the **stdio** header and the **printf** function



```
#include <stdio.h>

int main()
{
    printf("It displays stuff...\n");
    return 0;
}
```

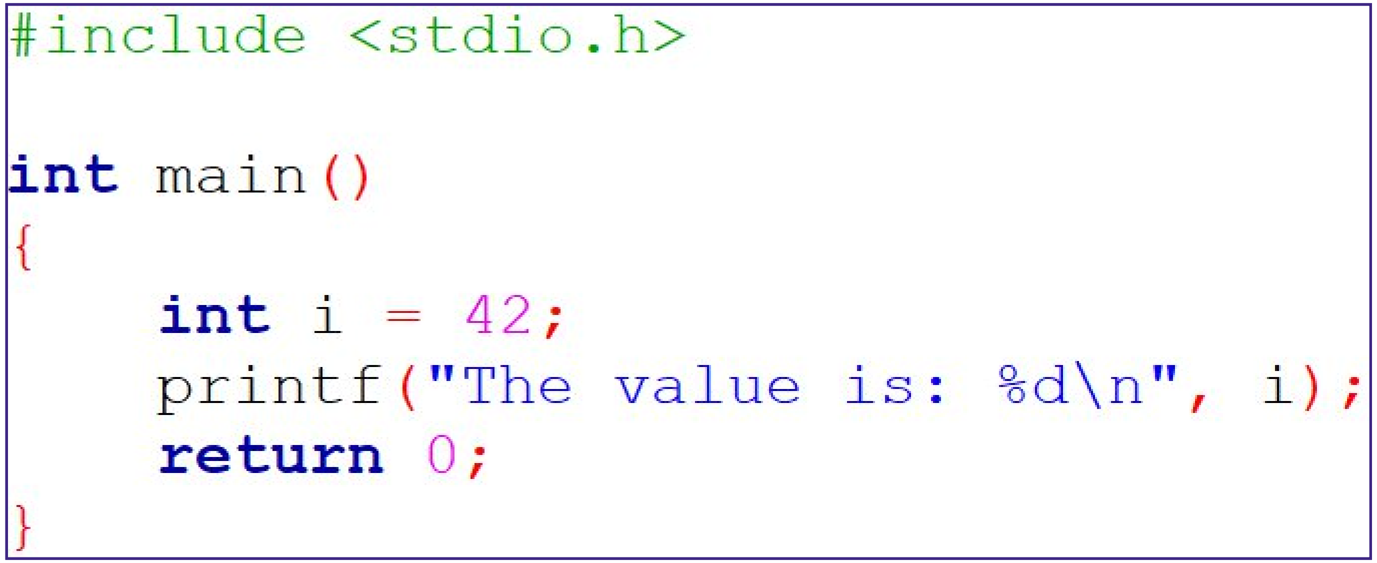


```
It displays stuff...
```

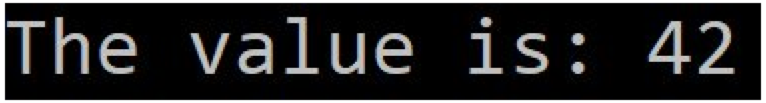
2. Variables

Display a variable

- To display variables, use a format specifier of variable type
  - **int** with the format specifier **%d**



```c
#include <stdio.h>

int main()
{
    int i = 42;
    printf("The value is: %d\n", i);
    return 0;
}
```

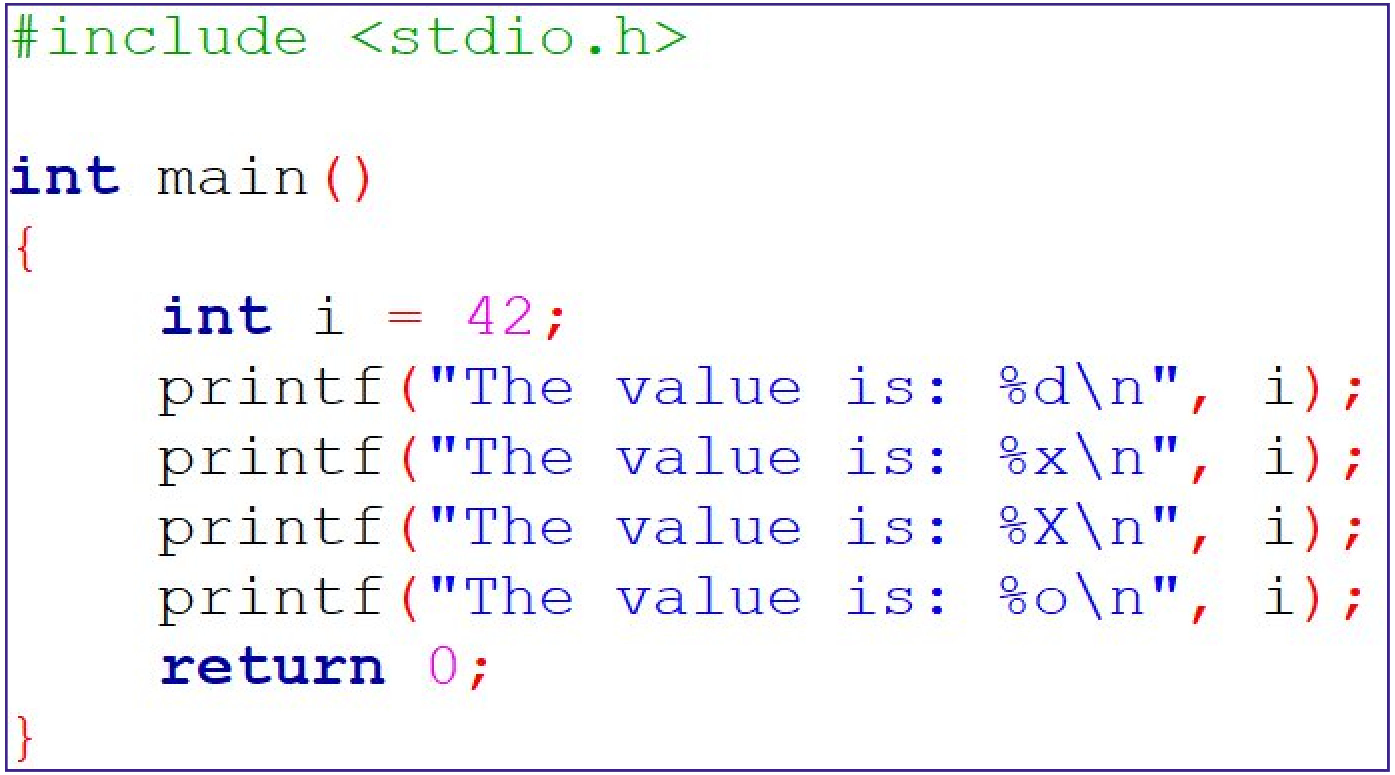


```
The value is: 42
```

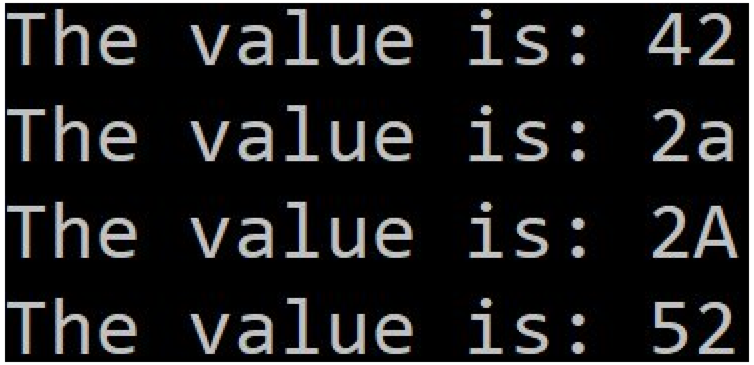
2. Variables

Display a variable

- **%x** or **%X** to convert in hexadecimal, **%o** in octal



```c
#include <stdio.h>

int main()
{
    int i = 42;
    printf("The value is: %d\n", i);
    printf("The value is: %x\n", i);
    printf("The value is: %X\n", i);
    printf("The value is: %o\n", i);
    return 0;
}
```

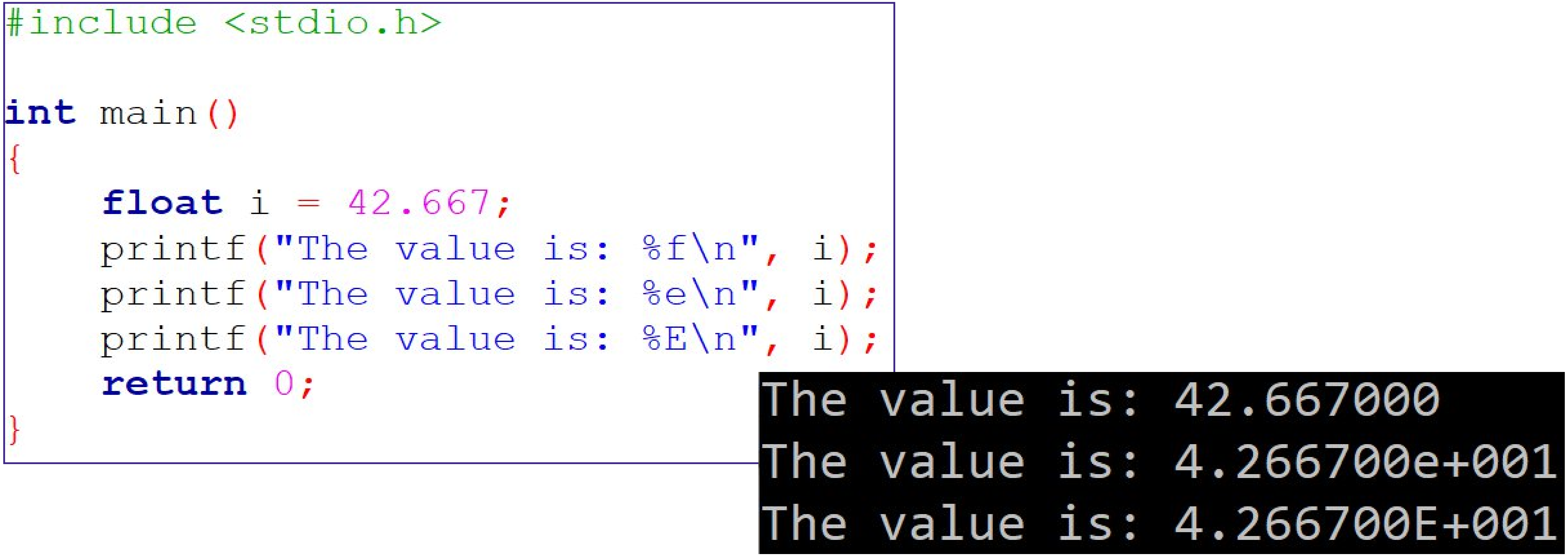


```
The value is: 42
The value is: 2a
The value is: 2A
The value is: 52
```

2. Variables

Display a variable

- **float** with the format specifier **%f** (**%e** or **%E** to use the scientific notation)



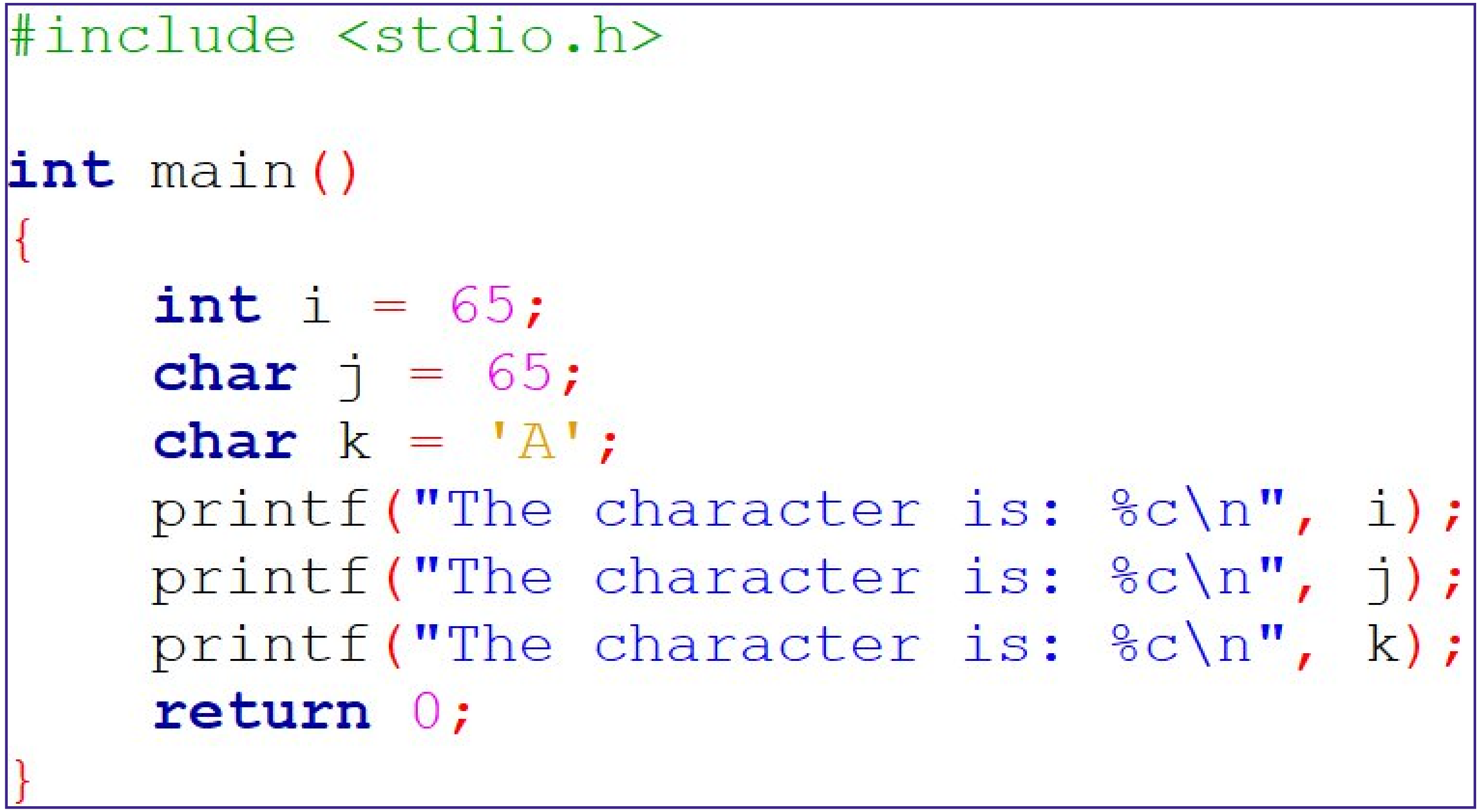
```c
#include <stdio.h>

int main()
{
    float i = 42.667;
    printf("The value is: %f\n", i);
    printf("The value is: %e\n", i);
    printf("The value is: %E\n", i);
    return 0;
}
```

```
The value is: 42.667000
The value is: 4.266700e+001
The value is: 4.266700E+001
```

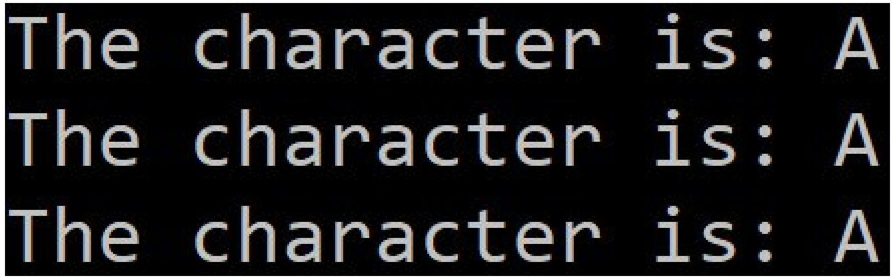
2. Variables

Display a variable

- **char** with the format specifier **%c**



```c
#include <stdio.h>

int main()
{
    int i = 65;
    char j = 65;
    char k = 'A';
    printf("The character is: %c\n", i);
    printf("The character is: %c\n", j);
    printf("The character is: %c\n", k);
    return 0;
}
```

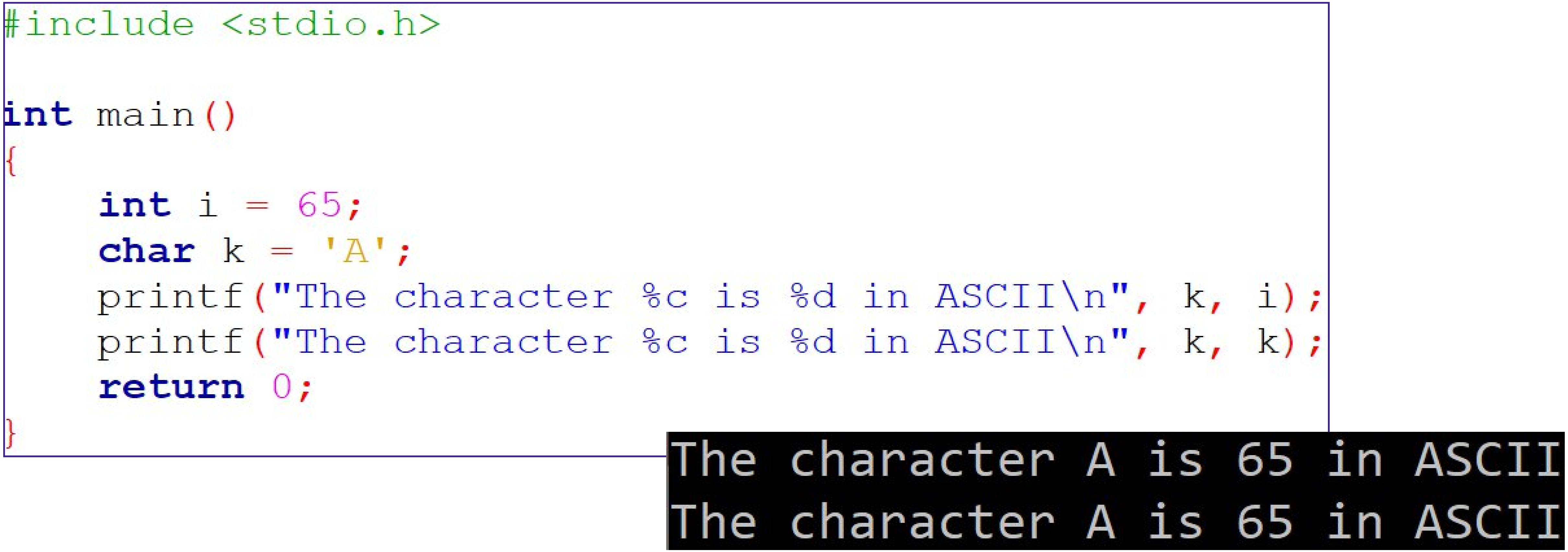


```
The character is: A
The character is: A
The character is: A
```

2. Variables

Display a variable

- You can display multiple variables with the same **printf**



```c
#include <stdio.h>

int main()
{
    int i = 65;
    char k = 'A';
    printf("The character %c is %d in ASCII\n", k, i);
    printf("The character %c is %d in ASCII\n", k, k);
    return 0;
}
```

```
The character A is 65 in ASCII
The character A is 65 in ASCII
```

2. Variables

Format specifier examples

Data Type	Format Specifier
char	%c
unsigned char	%c
short	%hd
unsigned short	%hu
int	%d
unsigned int	%u
long	%ld
unsigned long	%lu
long long	%lld
unsigned long long	%llu
float	%f
double	%lf
long double	%Lf
string	%s

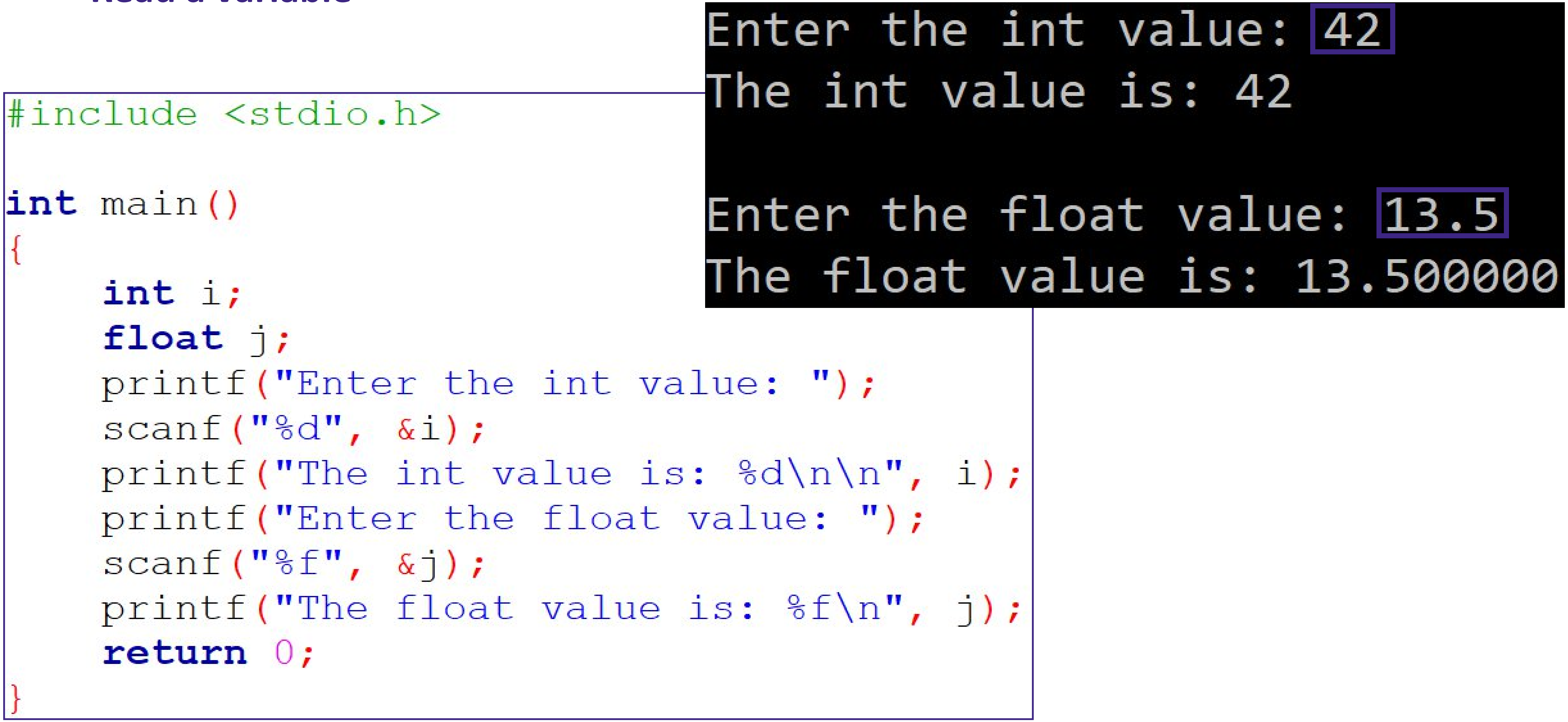
2. Variables

Read a variable

- Use the **stdio** header and the **scanf** function to request an input
- Use a format specifier of variable type
- Specify the address of the variable in the computer's memory
- To display text related to the request, you need to use **printf**

2. Variables

Read a variable



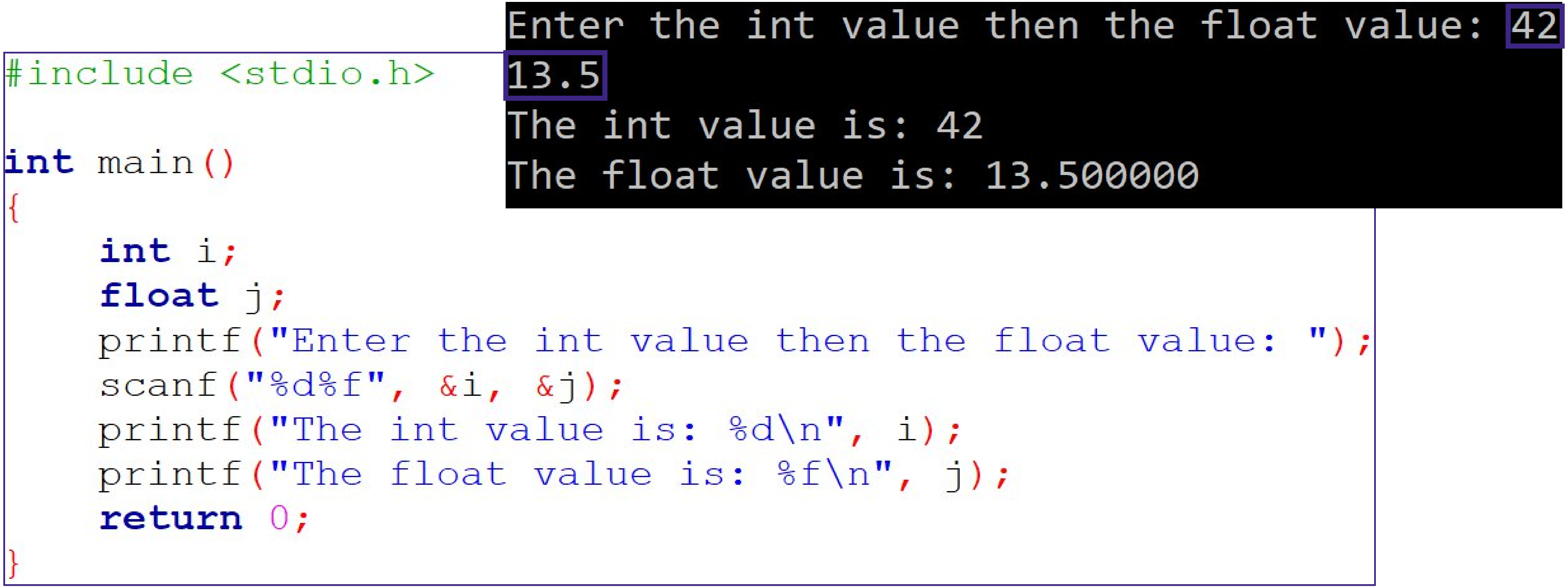
```
Enter the int value: 42
The int value is: 42

Enter the float value: 13.5
The float value is: 13.500000
```

```c
#include <stdio.h>

int main()
{
    int i;
    float j;
    printf("Enter the int value: ");
    scanf("%d", &i);
    printf("The int value is: %d\n\n", i);
    printf("Enter the float value: ");
    scanf("%f", &j);
    printf("The float value is: %f\n", j);
    return 0;
}
```

2. Variables

Read a variable

- You can request multiple variables with the same **scanf**



```c
#include <stdio.h>

int main()
{
    int i;
    float j;
    printf("Enter the int value then the float value: ");
    scanf("%d%f", &i, &j);
    printf("The int value is: %d\n", i);
    printf("The float value is: %f\n", j);
    return 0;
}
```

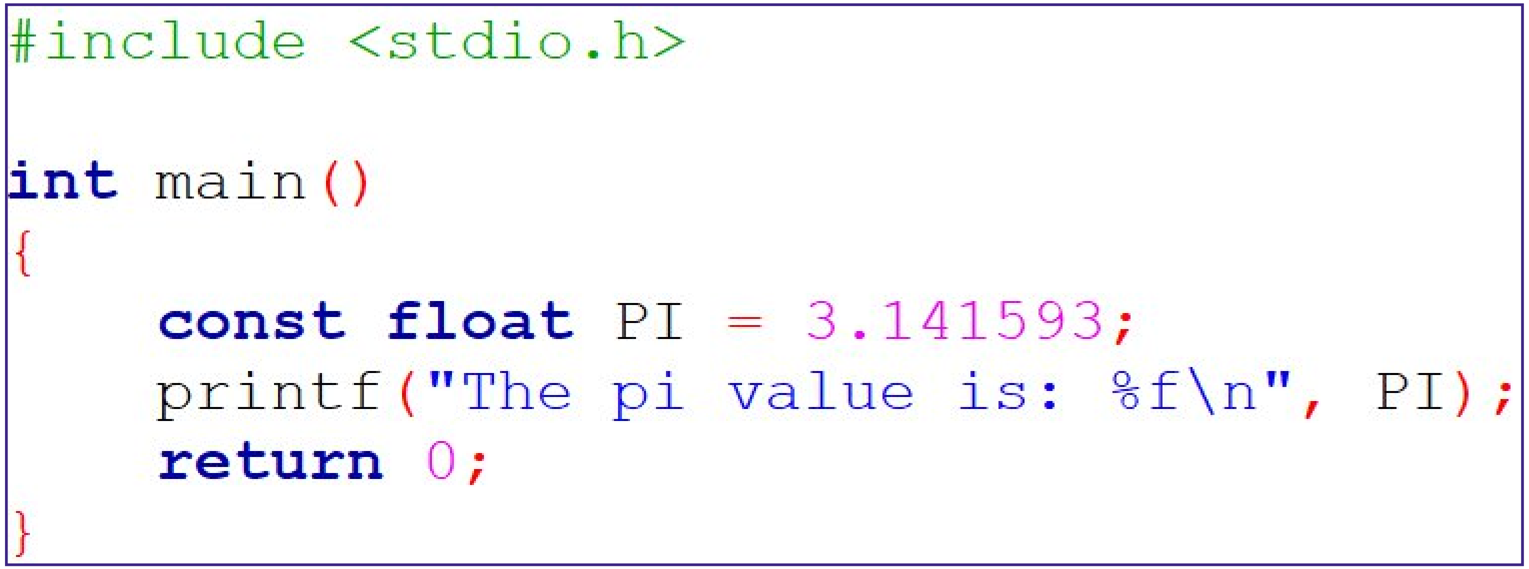
```
Enter the int value then the float value: 42
13.5
The int value is: 42
The float value is: 13.500000
```

2. Variables

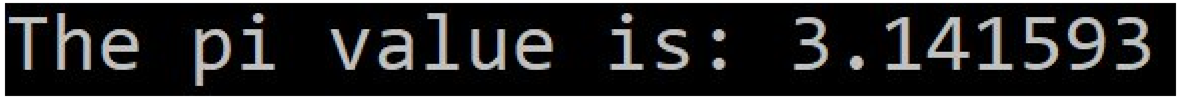
Constants

- Use the **const** keyword

- Do the assignment immediately



```c
#include <stdio.h>

int main()
{
    const float PI = 3.141593;
    printf("The pi value is: %f\n", PI);
    return 0;
}
```



```
The pi value is: 3.141593
```

2. Variables

Exercise

- You have the following URL, but 3 parts are missing:

https://www.youtube.com/watch?v=_WAOx_OmR_

- Display the URL using the following variables at the missing places:

1. The character associated with the ASCII decimal value 51
2. The character associated with the ASCII hexadecimal value 4B
3. The ASCII decimal value associated with the character Z

2. Variables

Questions

3. Preprocessor Directives

3. Preprocessor Directives

#include

Both user and system header files are included

- **#include <file>**
  - It is used for system header files
  - It searches for a file named **file** in directories pre-designated by the compiler/IDE (standard system)
  - It is normally used to include standard library header files

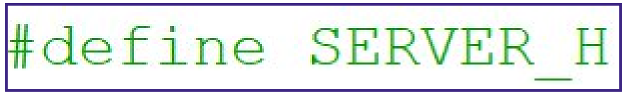
- **#include "file"**
  - It is used for header files of your own program
  - It searches for a file named **file** in the same directory as the file containing the directive (or using the relative path) then in pre-designated directories
  - It is normally used to include programmer-defined header files

3. Preprocessor Directives

#define

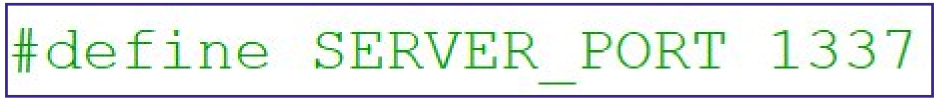
Define a constant or create a macro

- Define



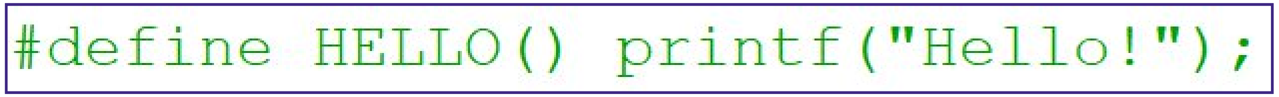
```
#define SERVER_H
```

- Constant



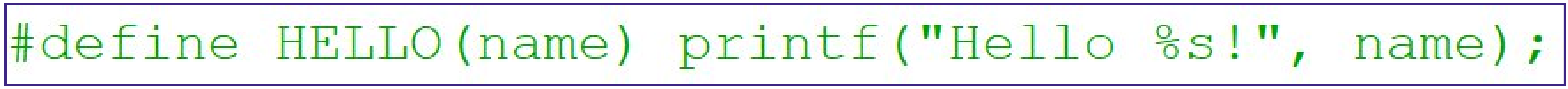
```
#define SERVER_PORT 1337
```

- Macro without argument



```
#define HELLO() printf("Hello!");
```

- Macro with argument(s)



```
#define HELLO(name) printf("Hello %s!", name);
```

3. Preprocessor Directives

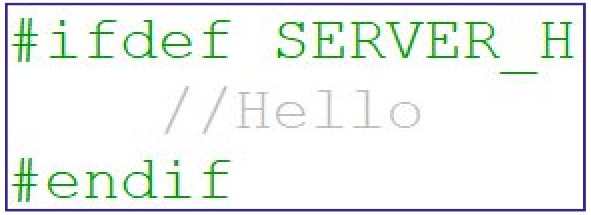
Predefined macros

- **__LINE__**: current line number
- **__FILE__**: current file full path
- **__DATE__**: current (compilation) date
- **__TIME__**: current (compilation) time
- **__TIMESTAMP__ ≈ __DATE__ + __TIME__**

3. Preprocessor Directives

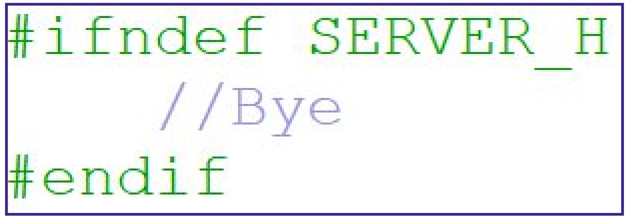
Conditions

- Check if **SERVER_H** is defined



```
#ifdef SERVER_H
    //Hello
#endif
```

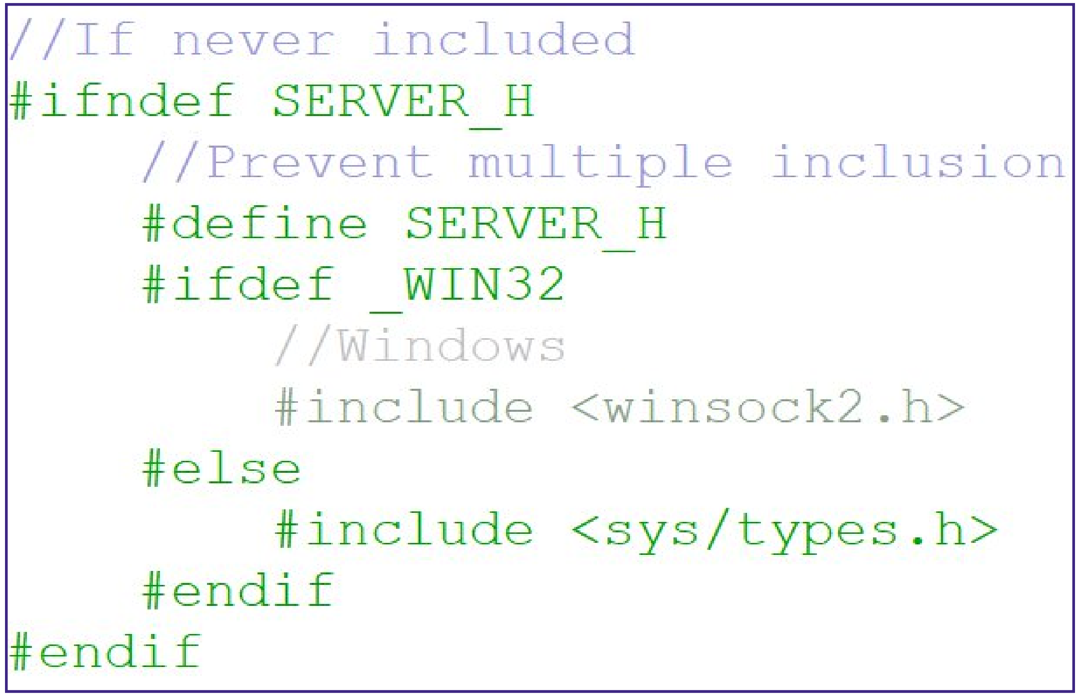
- Check if **SERVER_H** is not defined



```
#ifndef SERVER_H
    //Bye
#endif
```

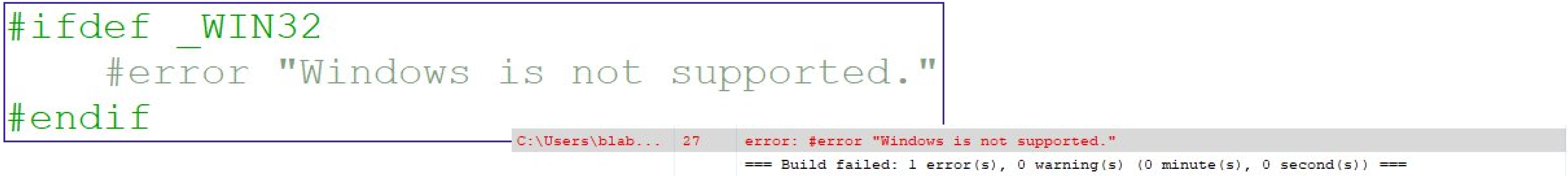
3. Preprocessor Directives

Conditions



```
//If never included
#ifndef SERVER_H
    //Prevent multiple inclusion
    #define SERVER_H
    #ifdef _WIN32
        //Windows
        #include <winsock2.h>
    #else
        #include <sys/types.h>
    #endif
#endif
```

- The **#error** macro allows you to make compilation fail and issue a statement that will appear in the list of compilation errors



```
#ifdef _WIN32
    #error "Windows is not supported."
#endif
```

```
C:\Users\blab...   27   error: #error "Windows is not supported."
                        === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) ===
```

3. Preprocessor Directives

Questions

4. Operators

4. Operators

Categories

- Arithmetic
- Assignment and incrementation
- Relational
- Logical
- Conditional

4. Operators

Arithmetic

- Arithmetic operators: **+**, **-**, *****, **/**, **%**

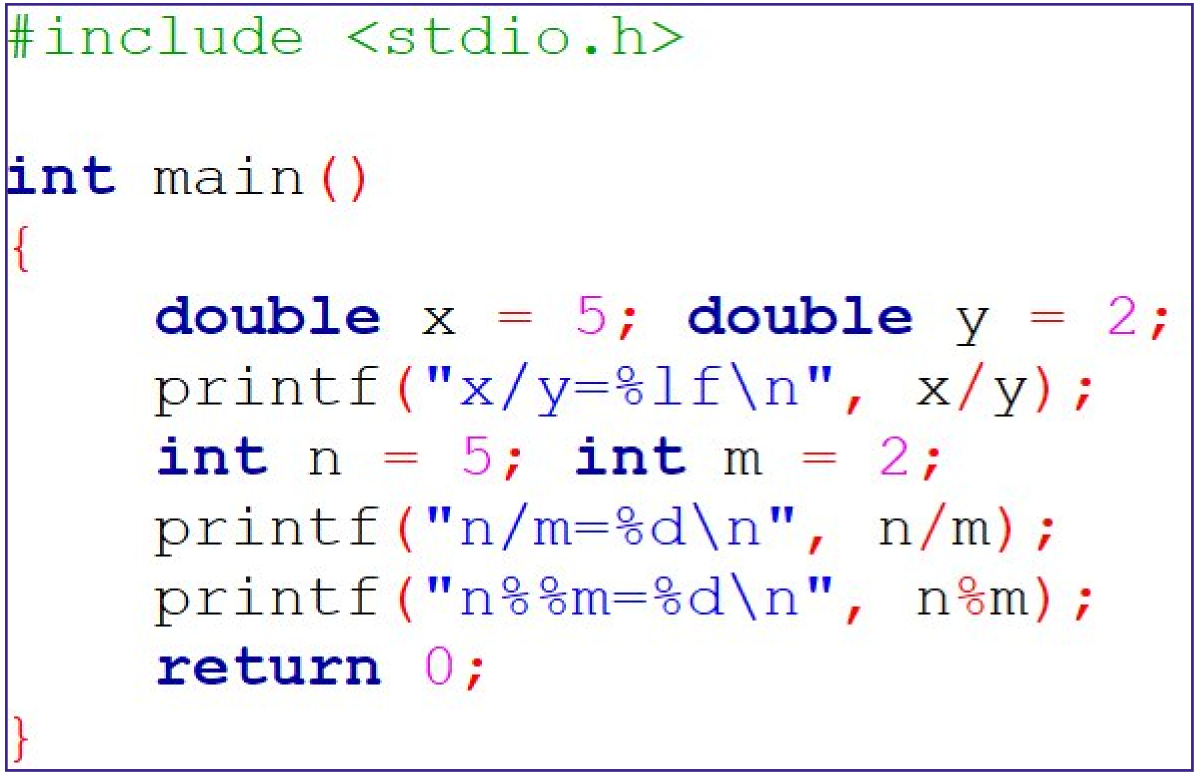
- They are usually defined only for operands of the same type

- When “**/**” is used with two integers, it returns the quotient of the Euclidean division of the first by the second

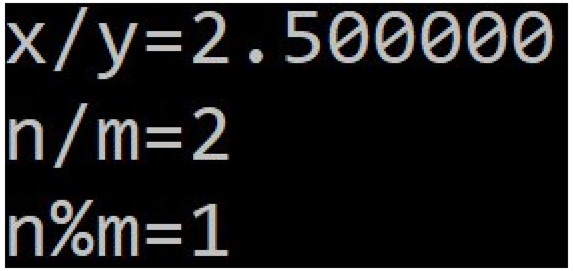
- The “**%**” operator is only defined with integers and returns the remainder of the Euclidean division of the first by the second

4. Operators

Arithmetic



```c
#include <stdio.h>

int main()
{
    double x = 5; double y = 2;
    printf("x/y=%lf\n", x/y);
    int n = 5; int m = 2;
    printf("n/m=%d\n", n/m);
    printf("n%%m=%d\n", n%m);
    return 0;
}
```



```
x/y=2.500000
n/m=2
n%m=1
```

4. Operators

Arithmetic

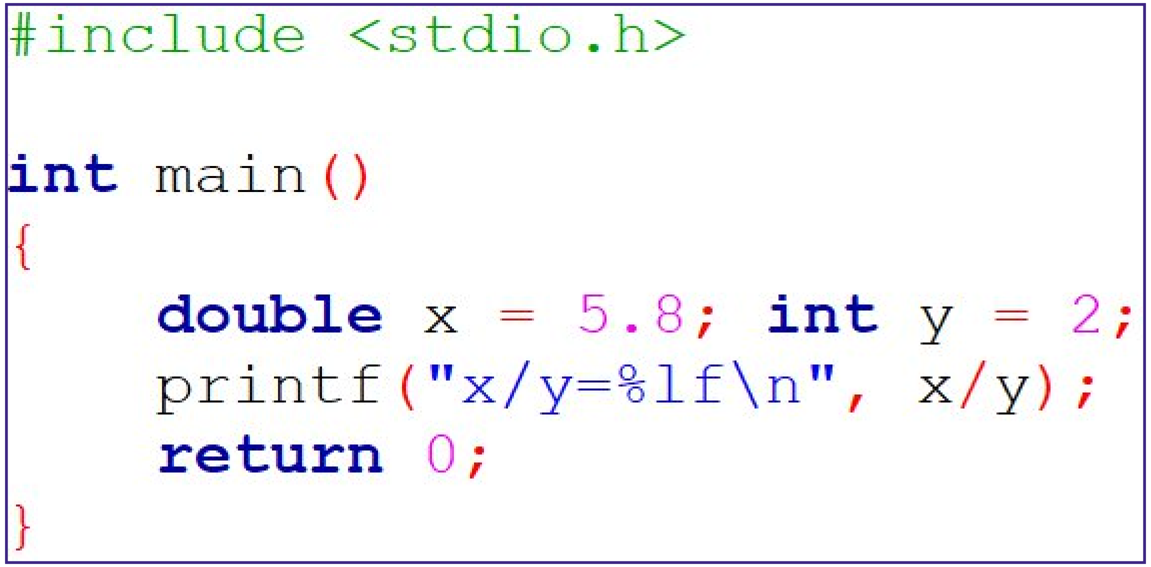
Reminders about implicit conversions:

- It may happen that we must perform a calculation between a real and an integer; the latter will then be implicitly converted into a real, and the result of the operation will also be real

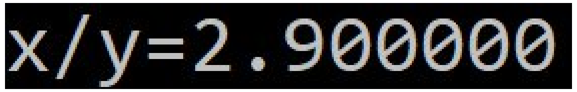
- More generally, if needed, a variable of a given type can be converted into a “higher” type (float into double for example, and not the other way around)

4. Operators

Arithmetic



```c
#include <stdio.h>

int main()
{
    double x = 5.8; int y = 2;
    printf("x/y=%lf\n", x/y);
    return 0;
}
```

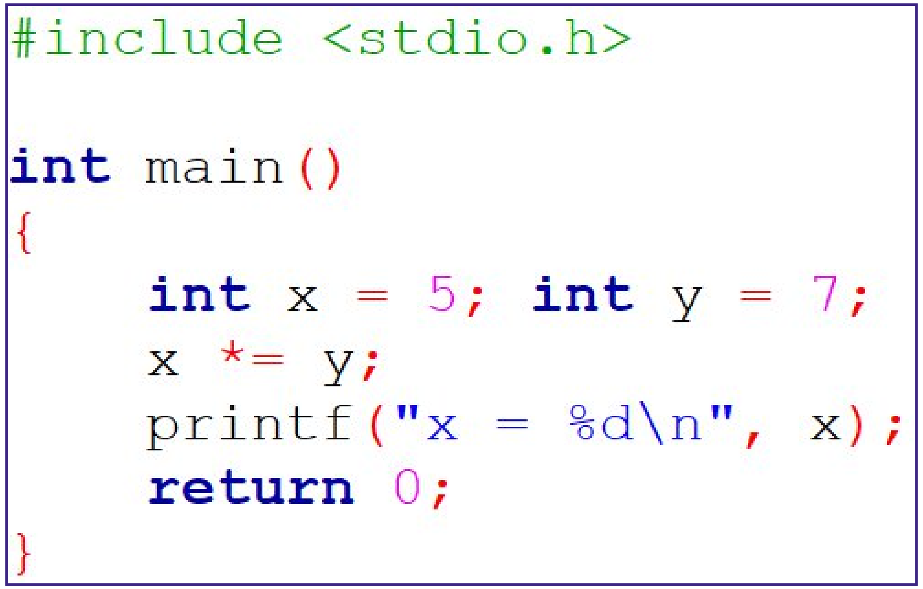


```
x/y=2.900000
```

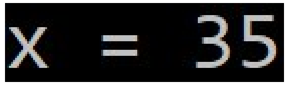
4. Operators

Assignment and incrementation

Assignment shortcuts	
x = x + y	x += y
x = x - y	x -= y
x = x * y	x *= y
x = x / y	x /= y
x = x % y	x %= y



```c
#include <stdio.h>

int main()
{
    int x = 5; int y = 7;
    x *= y;
    printf("x = %d\n", x);
    return 0;
}
```



```
x = 35
```

4. Operators

Assignment and incrementation

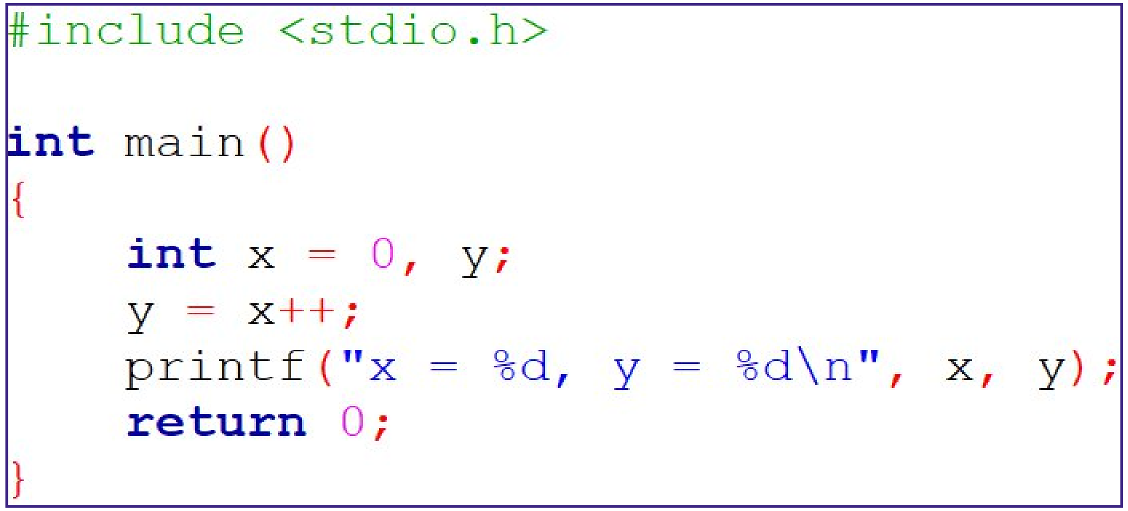
- To increment or decrement the value of a variable by 1 you can use the operators "**++**" or "**--**"

- **n++** and **++n** will thus increment n by 1 but the value of these expressions is however different: **n++** is the value of n before the increment and **++n** is the value of n after the increment

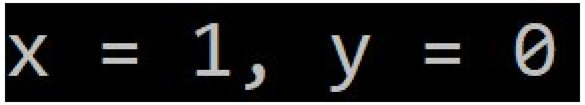
- Likewise with "**--**"

4. Operators

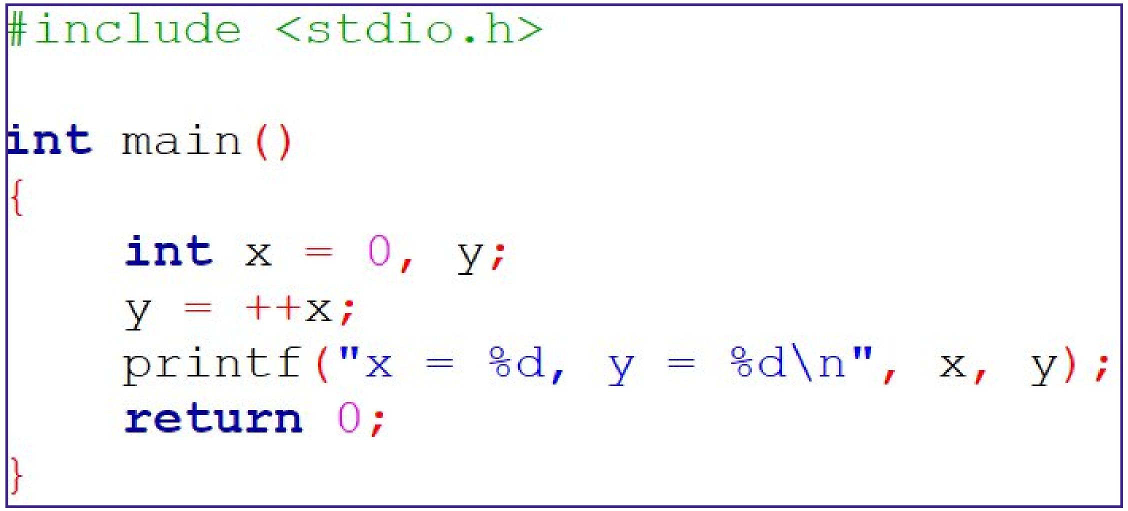
Assignment and incrementation



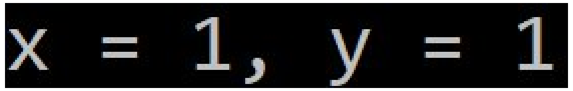
```c
#include <stdio.h>

int main()
{
    int x = 0, y;
    y = x++;
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```



```
x = 1, y = 0
```



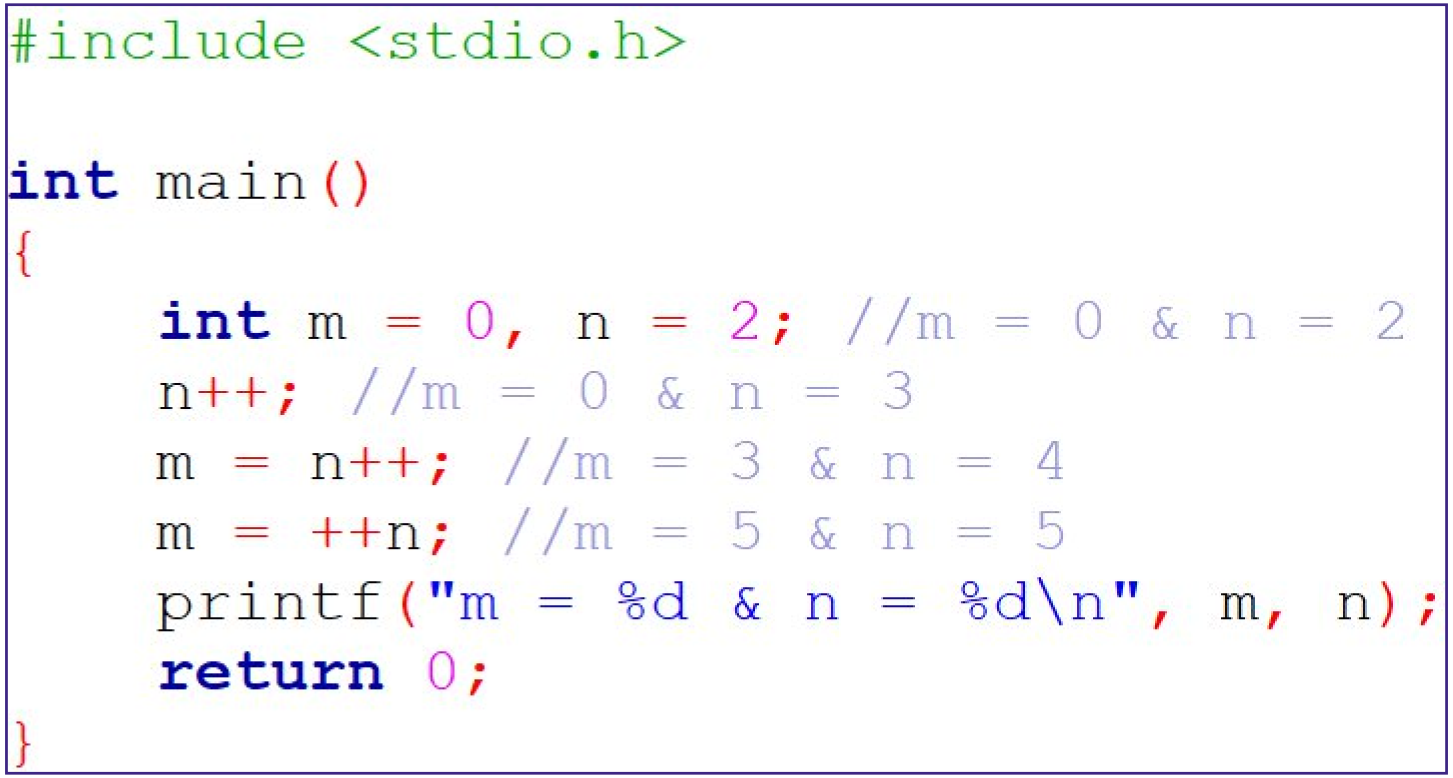
```c
#include <stdio.h>

int main()
{
    int x = 0, y;
    y = ++x;
    printf("x = %d, y = %d\n", x, y);
    return 0;
}
```



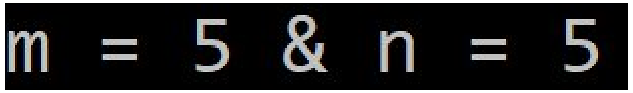
```
x = 1, y = 1
```

4. Operators

Assignment and incrementation



```c
#include <stdio.h>

int main()
{
    int m = 0, n = 2; //m = 0 & n = 2
    n++; //m = 0 & n = 3
    m = n++; //m = 3 & n = 4
    m = ++n; //m = 5 & n = 5
    printf("m = %d & n = %d\n", m, n);
    return 0;
}
```



```
m = 5 & n = 5
```

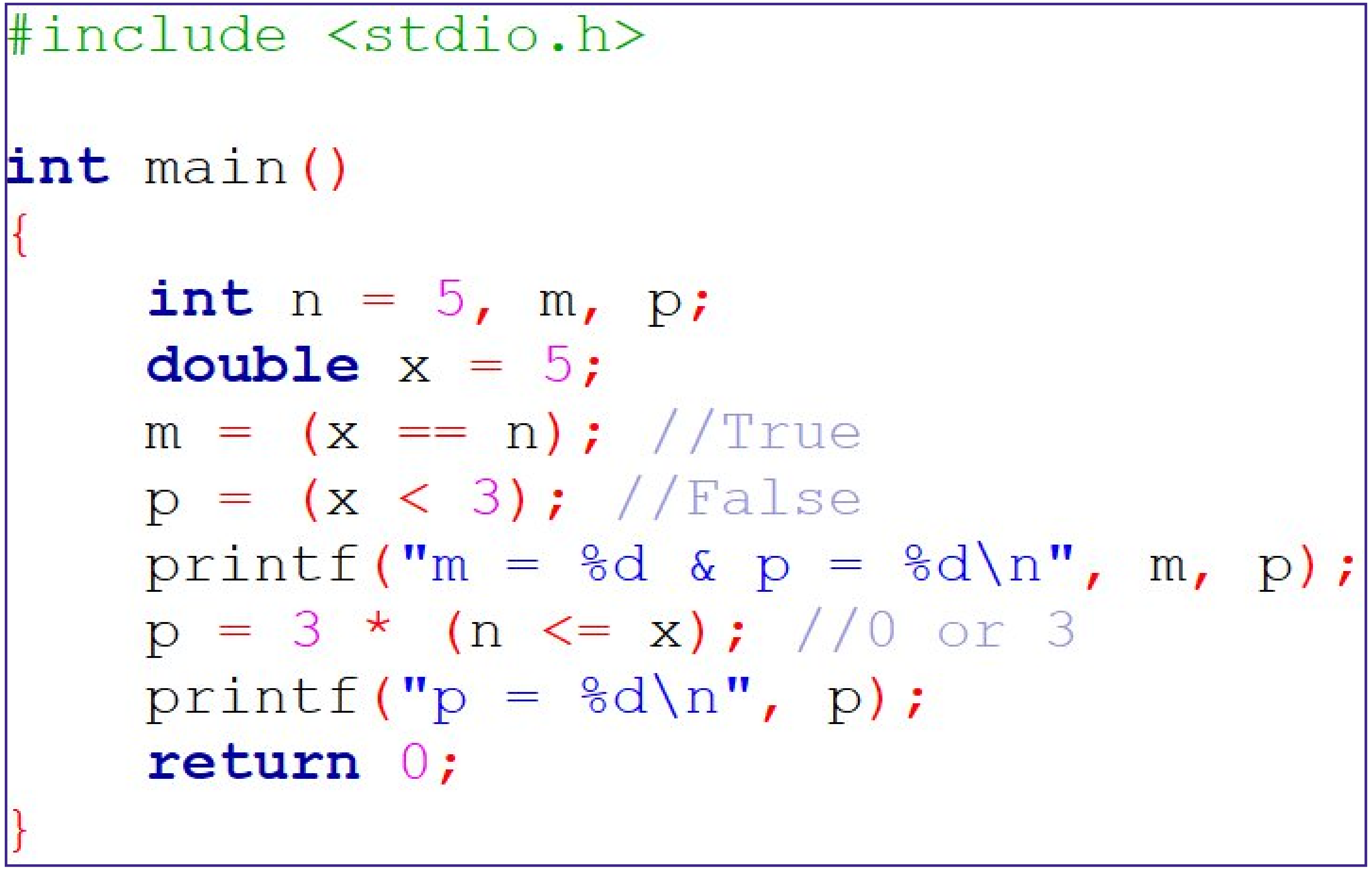
4. Operators

Relational

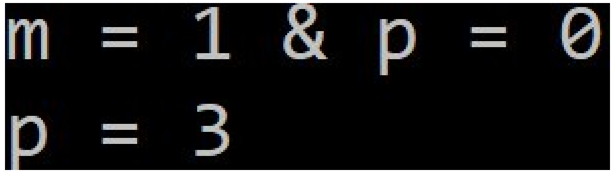
- Relational operators: **==**, **!=**, **<=**, **>=**, **<**, **>**
- They deal with numerical values
- They are subject to the implicit conversion rules
- The result is an integer, worth **0** if the comparison is false, and **1** if it is true

4. Operators

Relational



```
#include <stdio.h>

int main()
{
    int n = 5, m, p;
    double x = 5;
    m = (x == n); //True
    p = (x < 3); //False
    printf("m = %d & p = %d\n", m, p);
    p = 3 * (n <= x); //0 or 3
    printf("p = %d\n", p);
    return 0;
}
```



```
m = 1 & p = 0
p = 3
```

4. Operators

Logical

- Logical operators: **&&**, **||**, **!**

- They correspond to **AND**, **OR** and **NOT**

- They take as operand numerical values with the convention that 0 corresponds to **FALSE** and that any non-zero value corresponds to **TRUE**

- The result is an integer worth 0 (**FALSE**) or 1 (**TRUE**)

4. Operators

Logical – Truth tables

z = x && y

x	y	z
0	0	0
0	1	0
1	0	0
1	1	1

z = x || y

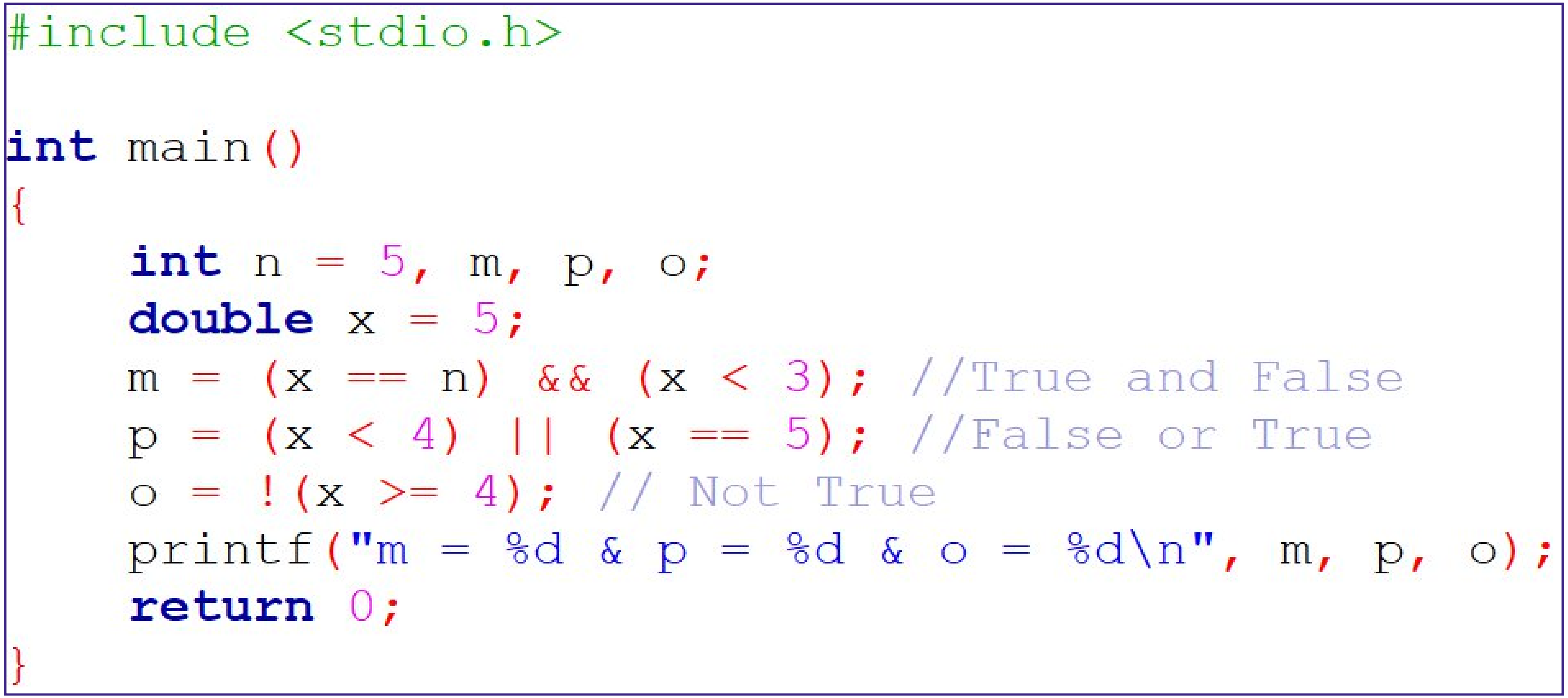
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

z = !x

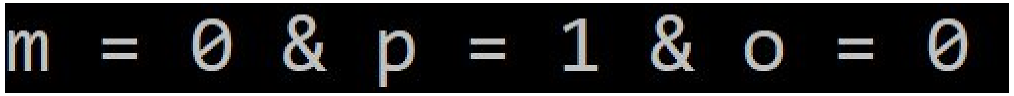
x	z
0	1
1	0

4. Operators

Logical



```c
#include <stdio.h>

int main()
{
     int n = 5, m, p, o;
     double x = 5;
     m = (x == n) && (x < 3); //True and False
     p = (x < 4) || (x == 5); //False or True
     o = !(x >= 4); // Not True
     printf("m = %d & p = %d & o = %d\n", m, p, o);
     return 0;
}
```



```
m = 0 & p = 1 & o = 0
```

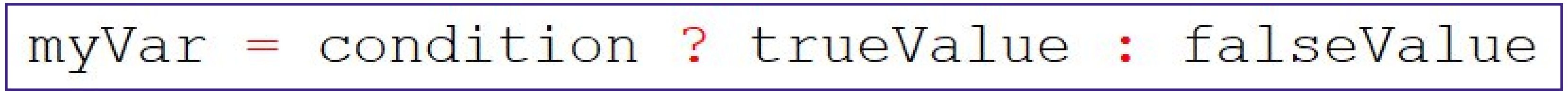
4. Operators

Conditional

- Conditional operator: **?**

- You can use the logical and relational operators

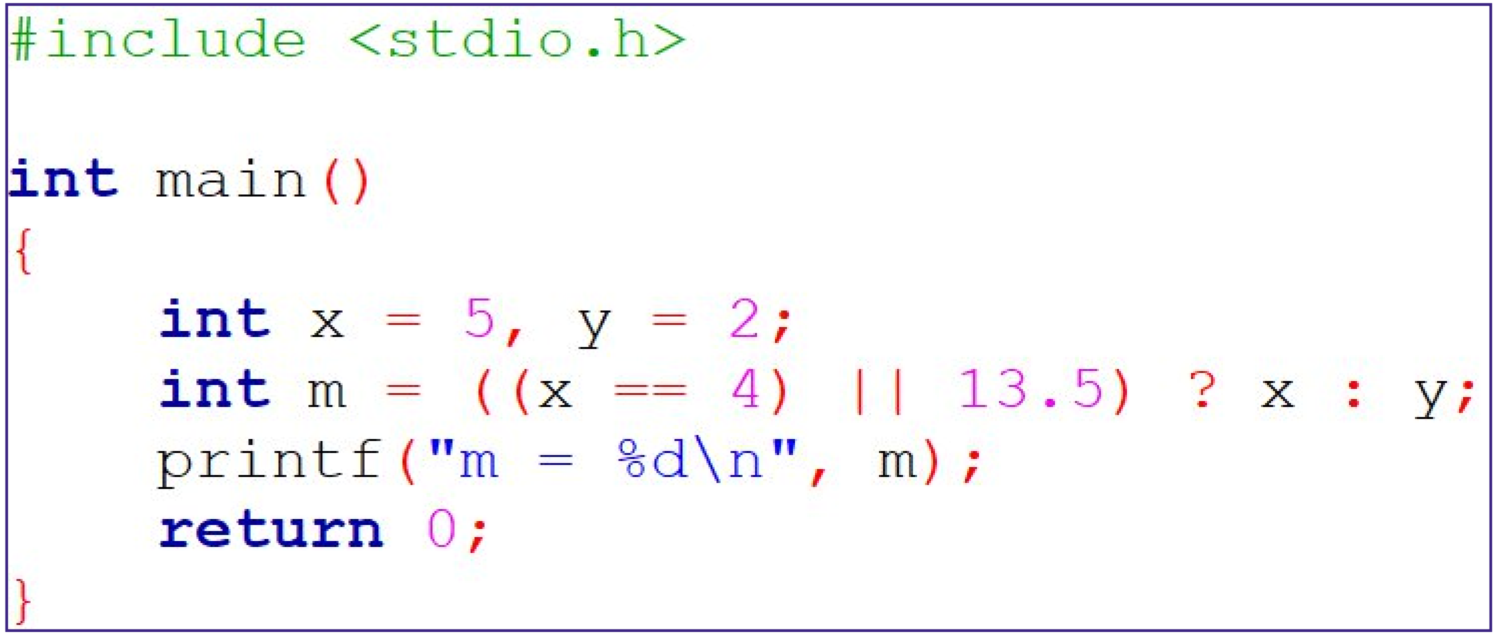
- Syntax:



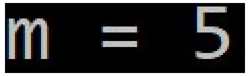
```
myVar = condition ? trueValue : falseValue
```

4. Operators

Conditional



```c
#include <stdio.h>

int main()
{
    int x = 5, y = 2;
    int m = ((x == 4) || 13.5) ? x : y;
    printf("m = %d\n", m);
    return 0;
}
```



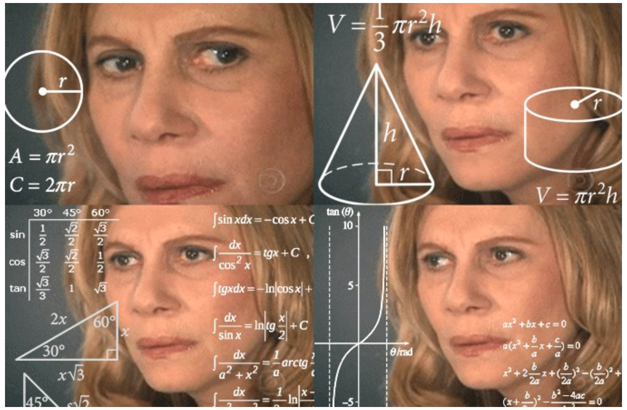
```
m = 5
```

4. Operators

Math functions

#include <math.h>

acos	asin	atan	atan2
cos	cosh	sin	sinh
tanh	exp	frexp	ldexp
log	log10	modf	pow
sqrt	ceil	fabs	floor
fmod			



4. Operators

Exercise

- Ask the user to enter a unit price before tax, a VAT rate and a quantity of items

- Calculate and display the total price including VAT of the purchase

4. Operators

Questions

C Developer

Discover the C Syntax

Thank you for your attention