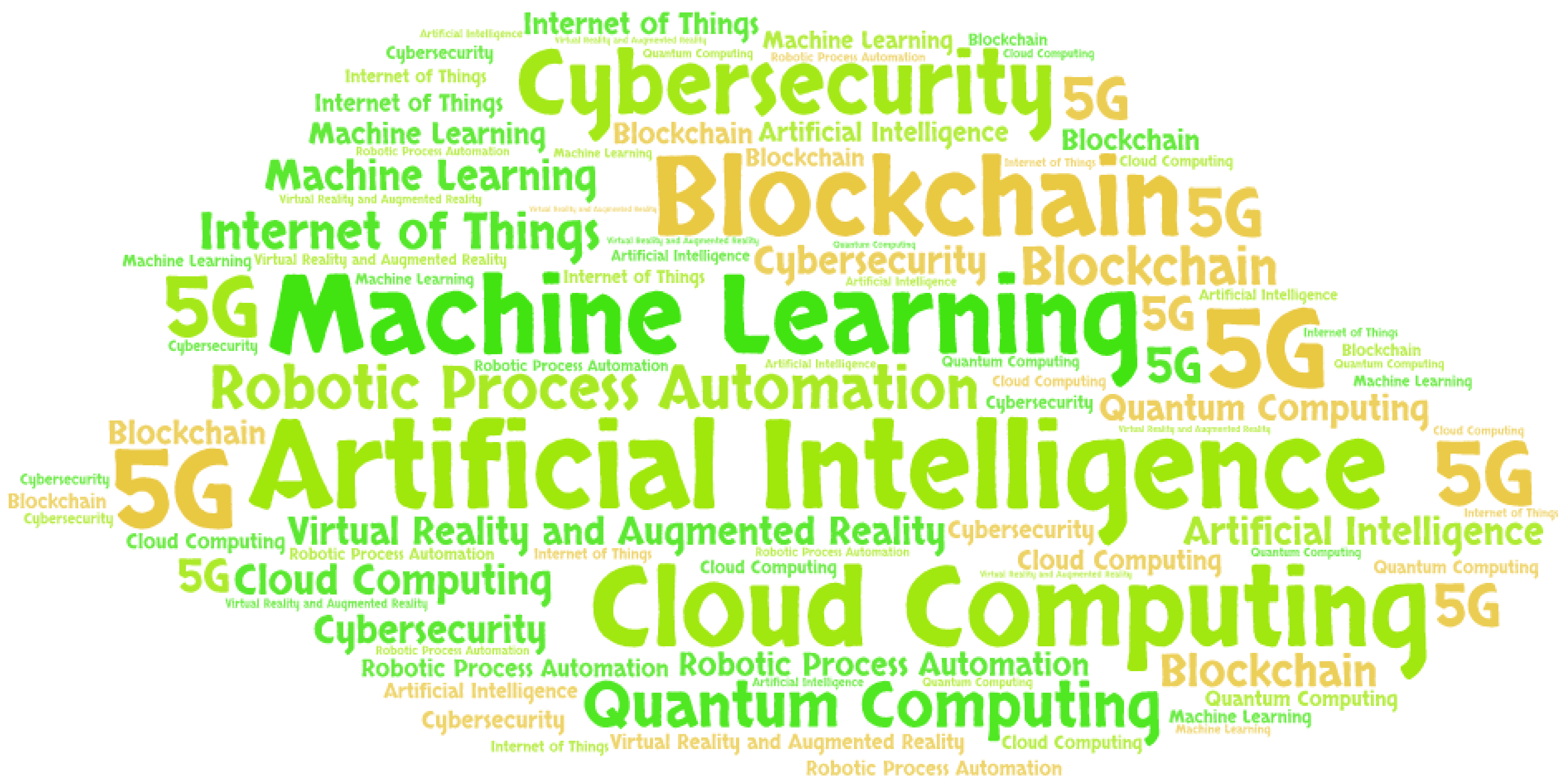
Welcome!

About Me

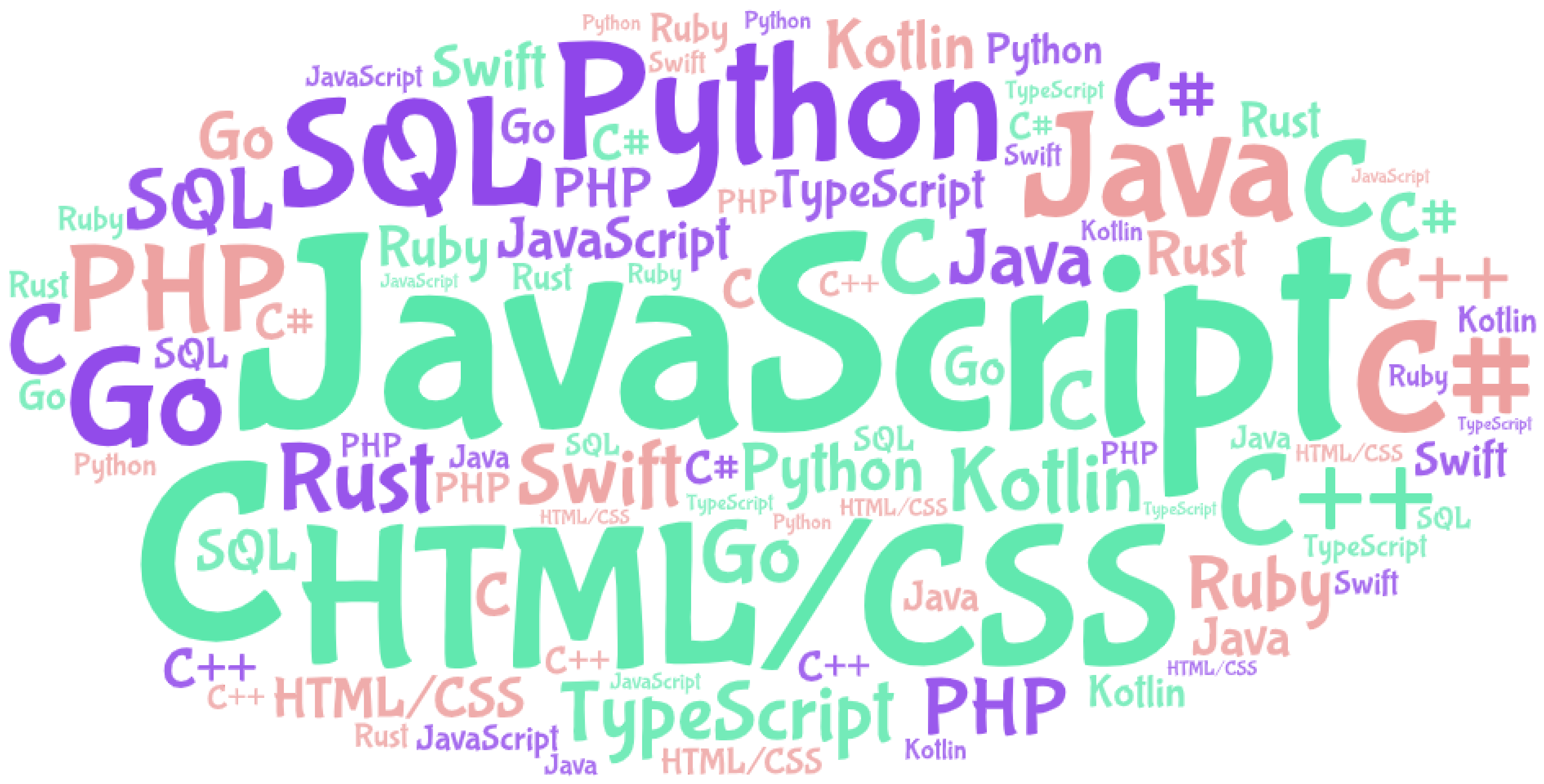
- Full-time software engineer in the field of cybersecurity
- Part-time Computer Science lecturer with CodeIT
- Our aim is to help students learn the fundamentals of Computer Science, and be able to apply what they learn to solve problems and pick up new software skills

CodeIT

Top trends in tech



Top programming languages



What you’ll learn in this course

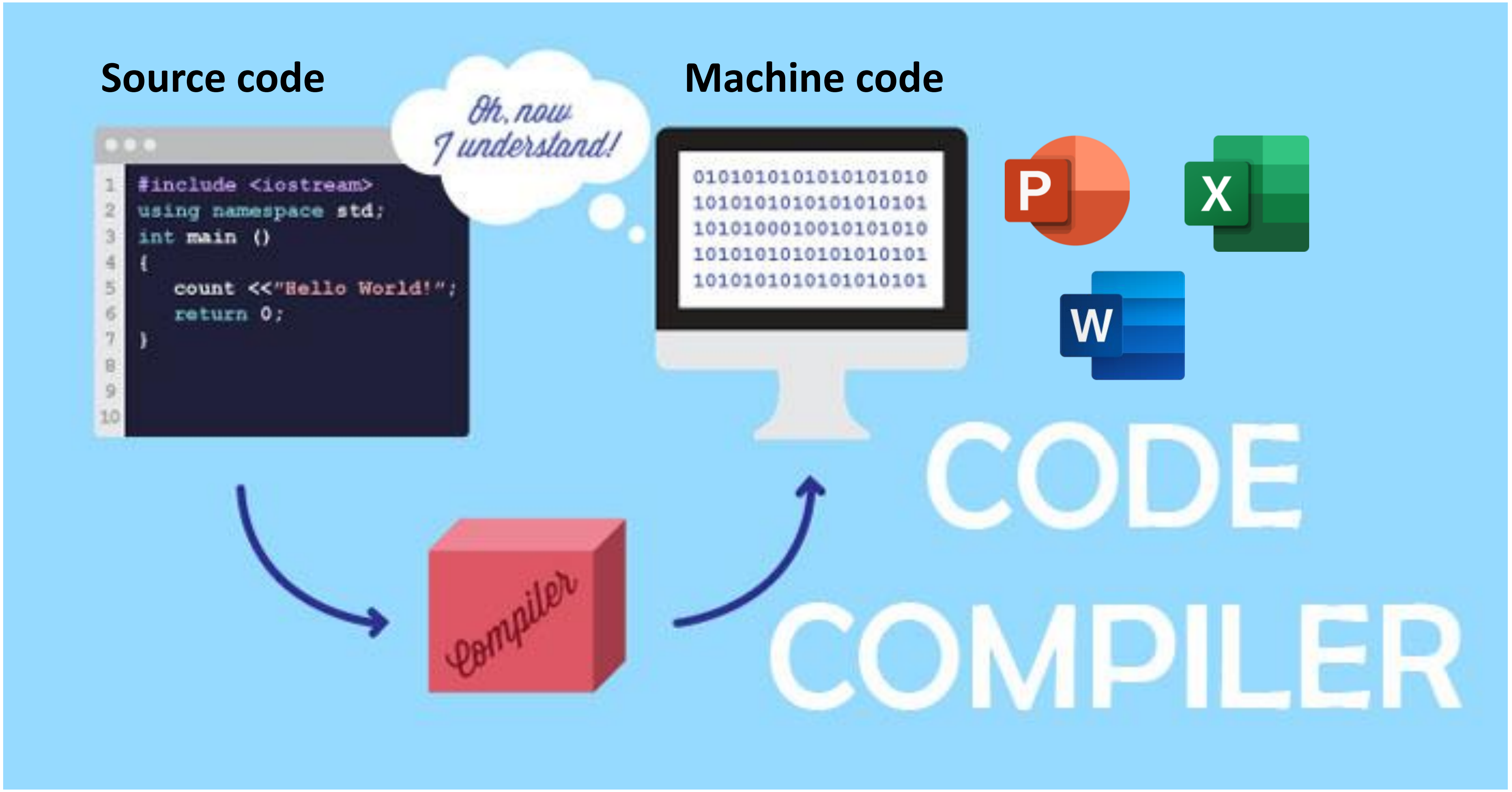
- Fundamentals of the C programming language
- Fundamentals of the Python programming language
- How to think and solve problems

THE C PROGRAMMING LANGUAGE

python™

What is a program?

- Computers are machines. They don’t think for themselves. They need us to tell them what to do.
- We need a way to tell our computers to do what we want them to, but computers don’t understand English.
- Writing code is a way for us to talk to our computers by giving them instructions.
- A special program, called a compiler, translates our code into a special language called machine code which computers can understand. Machine code is written in binary.

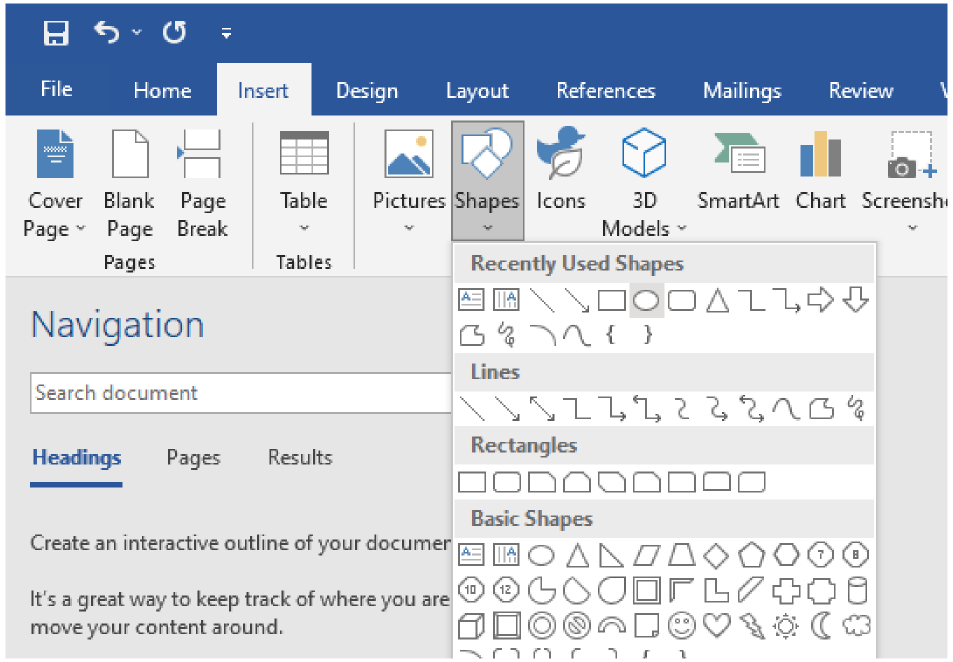


```
#include <iostream>
using namespace std;
int main ()
{
    count <<"Hello World!";
    return 0;
}
```

```
01010101010101010
10101010101010101
10101000100101010
10101010101010101
10101010101010101
```

Abstraction

- Abstraction is the process of hiding implementation details and only showing the functionality to the user.
- For example, when you wash your laundry in a laundry machine, you put your laundry and detergent inside the machine, press a button, and wait for the machine to perform its task. But how does the machine wash your clothes? What mechanism does it use? A user is not required to know the engineering behind its workings. This allows the user to efficiently use the laundry machine just with a click of the button.

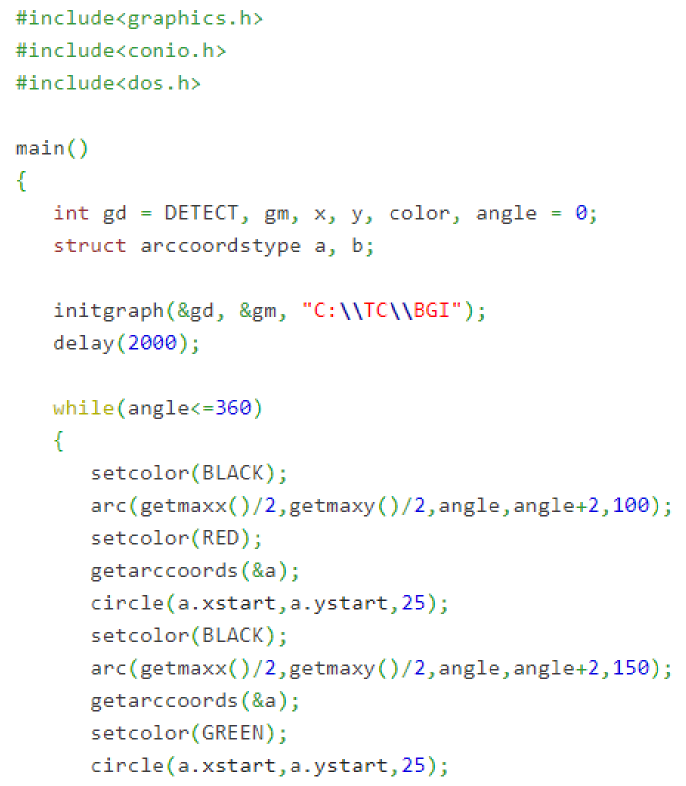


Create an interactive outline of your documen

It's a great way to keep track of where you are
move your content around.

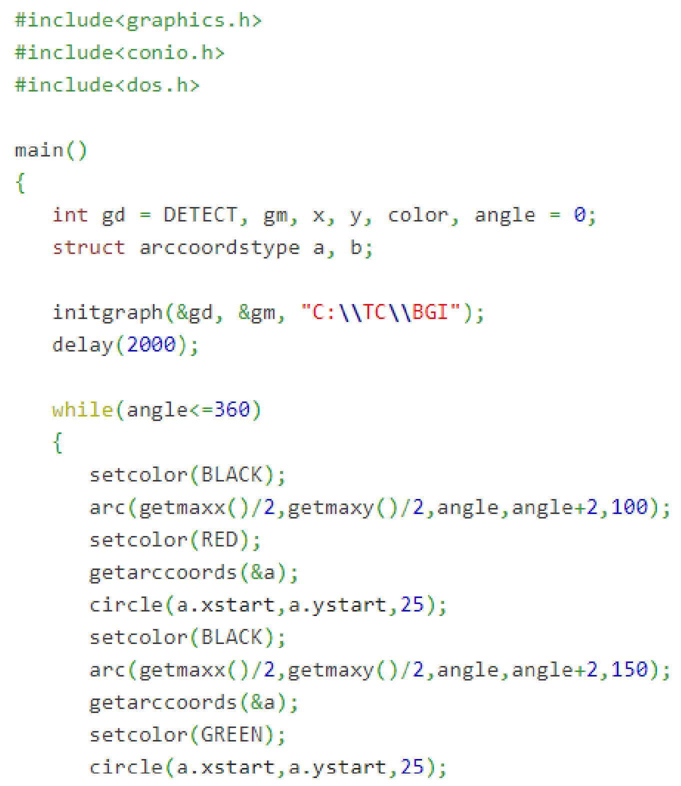
Click a button to create a circle shape in Microsoft Word. But how is this implemented?

Less abstract



```
#include<graphics.h>
#include<conio.h>
#include<dos.h>

main()
{
   int gd = DETECT, gm, x, y, color, angle = 0;
   struct arccoordstype a, b;

   initgraph(&gd, &gm, "C:\\TC\\BGI");
   delay(2000);

   while(angle<=360)
   {
      setcolor(BLACK);
      arc(getmaxx()/2,getmaxy()/2,angle,angle+2,100);
      setcolor(RED);
      getarccoords(&a);
      circle(a.xstart,a.ystart,25);
      setcolor(BLACK);
      arc(getmaxx()/2,getmaxy()/2,angle,angle+2,150);
      getarccoords(&a);
      setcolor(GREEN);
      circle(a.xstart,a.ystart,25);
```

C code for creating a circle. But how is this implemented?



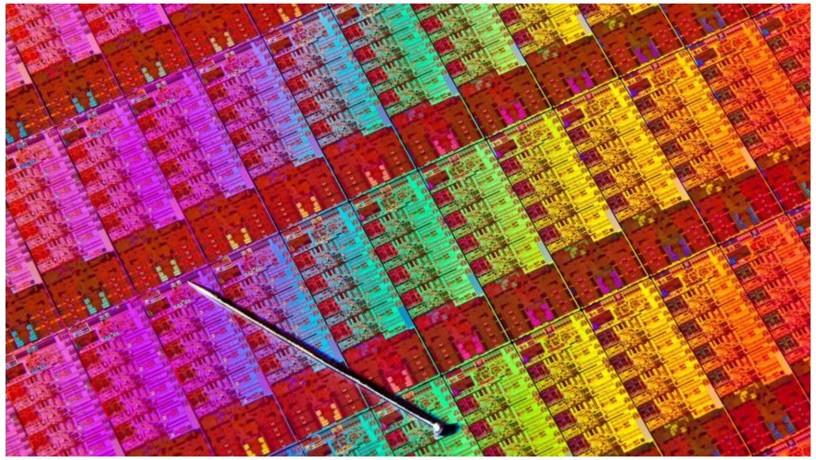
```
#include<graphics.h>
#include<conio.h>
#include<dos.h>

main()
{
   int gd = DETECT, gm, x, y, color, angle = 0;
   struct arccoordstype a, b;

   initgraph(&gd, &gm, "C:\\TC\\BGI");
   delay(2000);

   while(angle<=360)
   {
      setcolor(BLACK);
      arc(getmaxx()/2,getmaxy()/2,angle,angle+2,100);
      setcolor(RED);
      getarccoords(&a);
      circle(a.xstart,a.ystart,25);
      setcolor(BLACK);
      arc(getmaxx()/2,getmaxy()/2,angle,angle+2,150);
      getarccoords(&a);
      setcolor(GREEN);
      circle(a.xstart,a.ystart,25);
```

C code for creating a circle. But how is this implemented?

Less abstract

Machine code for creating the C code. But how is the machine code implemented? How does it get read by computers?

Machine code for creating the C code. But how is the machine code implemented? How does it get read by computers? At this stage, we are still working with software



At the lowest, most basic level, computers are made up of tiny transistors that work like switches that can be turned on or off. These switches are flipped on or off by an electrical charge. To tell a computer what to do, we send electric charges to flip different switches on or off. A one in machine code flips a switch on, while a zero flips the switch off. This is how computers understand machine code! At this point, we are working with hardware.

The C programming language

- Levels of abstraction – a lower level (less abstract) programming language is a lot closer to what machines understand, but more difficult for us humans to understand. A higher level (more abstract) programming language is closer to what we humans can understand, but needs to be converted more times in order to obtain machine code
- Machines can only understand one language – machine code. This is also the lowest level (least abstract) programming language. Sometimes you will hear about assembly language. This is a higher level language than machine code, but lower level than C
- The C language is lower-level than most other languages, including Python.
- However, sometimes you will hear that C is a high level language because it is still higher level than assembly language and machine code

The C programming language

- C programming helps you to understand better how computers work
- A foundation in C makes it easier to learn other programming languages
- C programming is very widely used. Here are some of them:
  - To program operating systems, like Windows and Linux – this is the software that powers your desktops, laptops, and mobile phones!
  - To program embedded systems – these include microcontrollers and processors that are used to power robots, IoT devices, automobiles, and many more!
  - To develop powerful applications, such as Microsoft Office programs, graphic design tools, computer games, and many more!

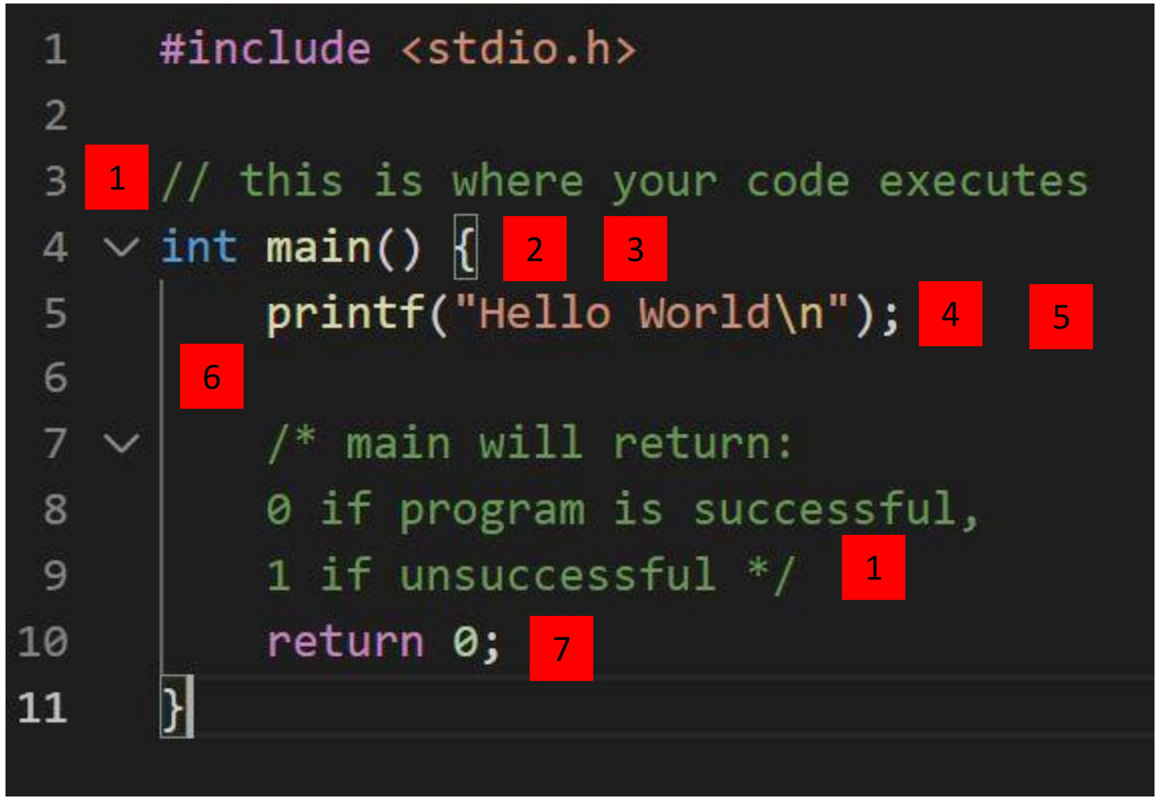
Steps of writing a program

1. Define the program objectives
2. Design the program
3. Write the code
4. Compile
5. Run the program
6. Test and debug the program

Your first Hello World program

- This is your first step to being a programmer
- In any new programming language we learn, the first thing we do is to learn how to show “Hello World” on our screen
- If you see the words “Hello World” appear on your screen, it means you have written your code correctly, and you are able to run your program!

Your first Hello World program



```
#include <stdio.h>

// this is where your code executes
int main() {
    printf("Hello World\n");

    /* main will return:
    0 if program is successful,
    1 if unsuccessful */
    return 0;
}
```

1 This is a comment. Comments should start with 2 forward slashes, or /* */. Comments are ignored and do not run like the rest of the code. They are only for people to read. Always make it a habit to write comments.

2 Every program *must* have a main() function. This is where the computer starts running the program. If you don't have a main function, your compiler cannot compile your code and you cannot run your code.

3 All functions should have an opening and closing curly braces

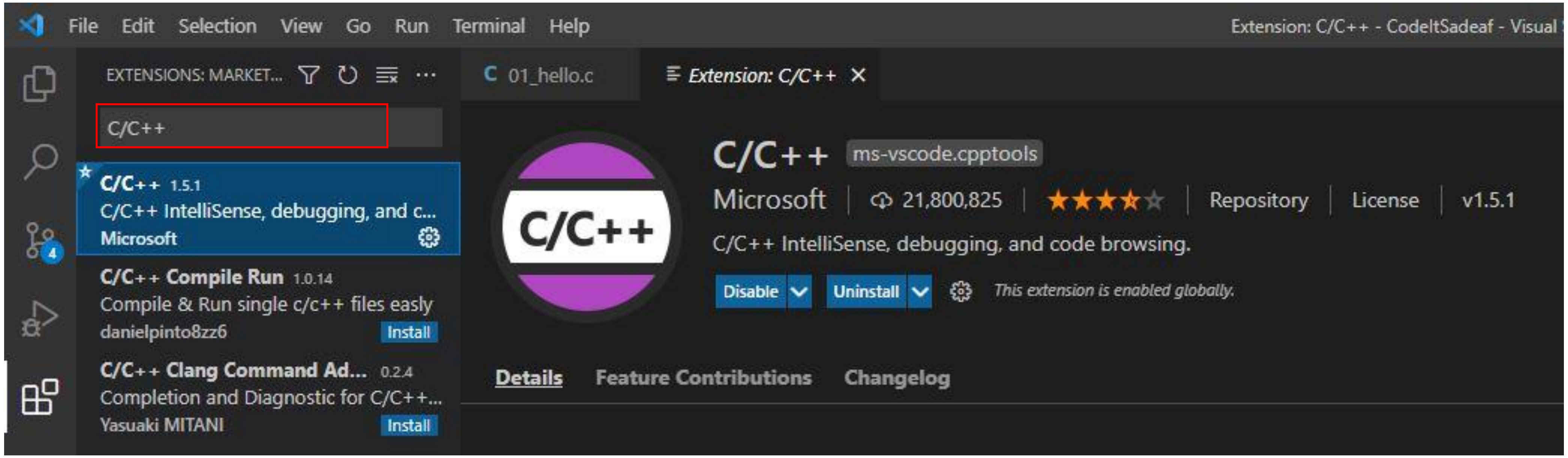
4 This is a built-in C function that lets you print (show) output onto screen

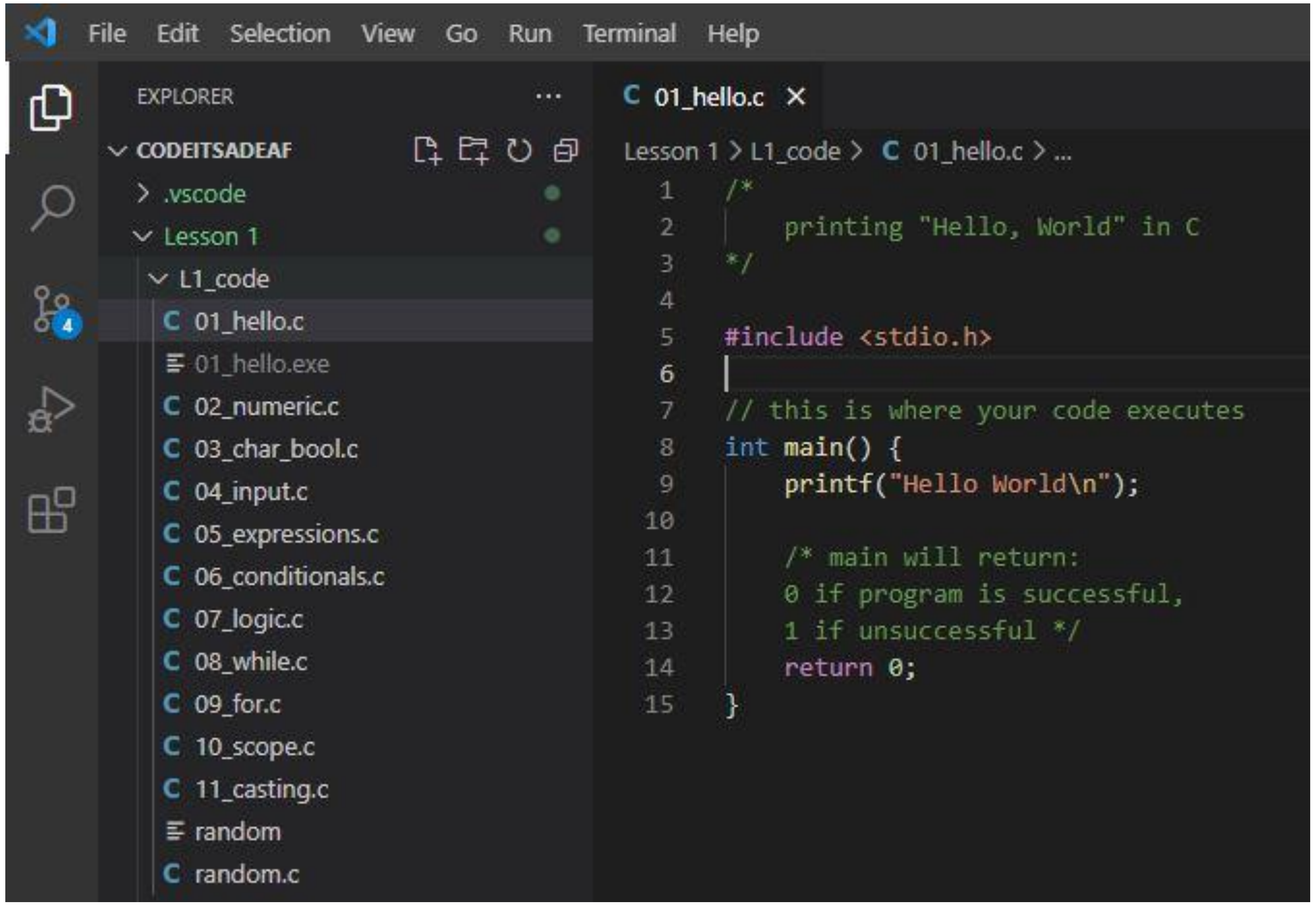
5 All statements should end with a semicolon.

6 Spaces, newlines, tabs, extra semicolons are ignored by the C compiler, but they are good for readability

7 The main() function must have a return statement. This returns the number 0 or 1 based on whether the program was run successfully or not

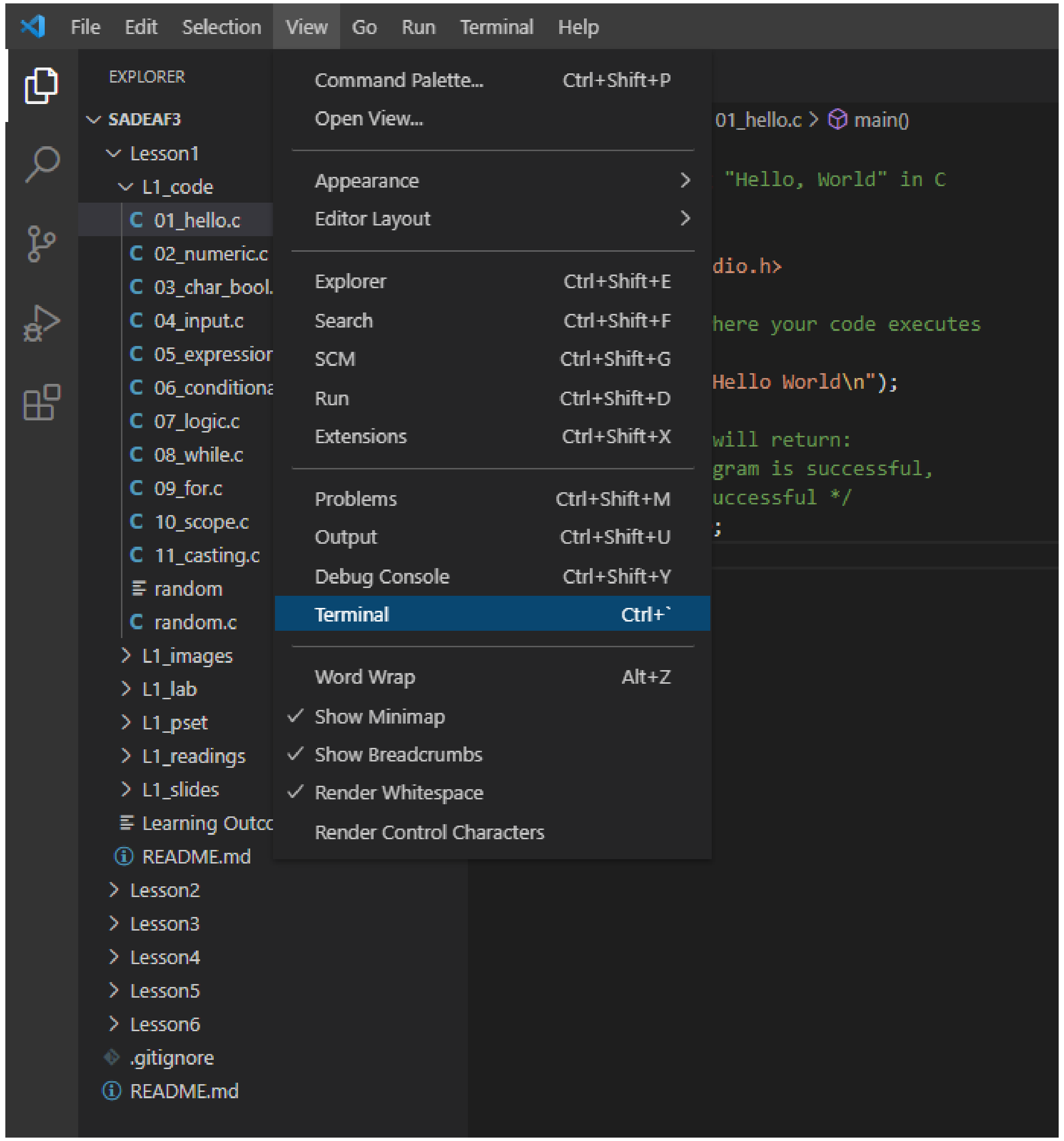
Install C/C++ extension





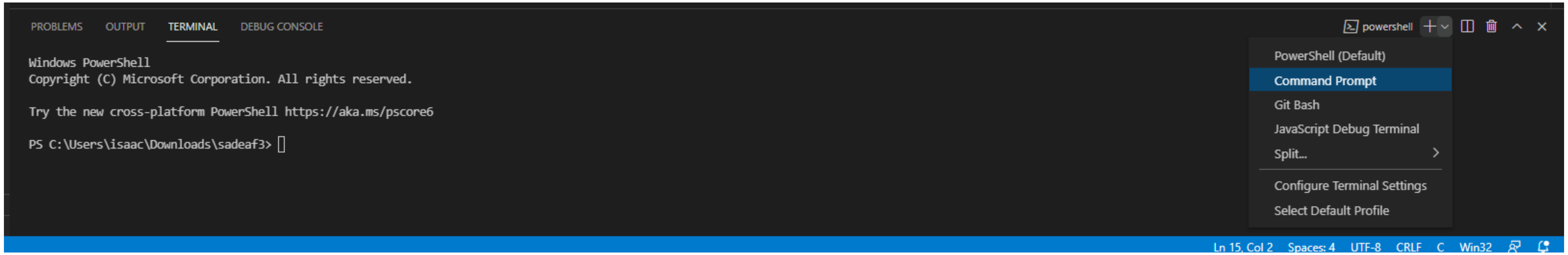
```c
/*
    printing "Hello, World" in C
*/

#include <stdio.h>

// this is where your code executes
int main() {
    printf("Hello World\n");

    /* main will return:
    0 if program is successful,
    1 if unsuccessful */
    return 0;
}
```

Open 01_hello.c



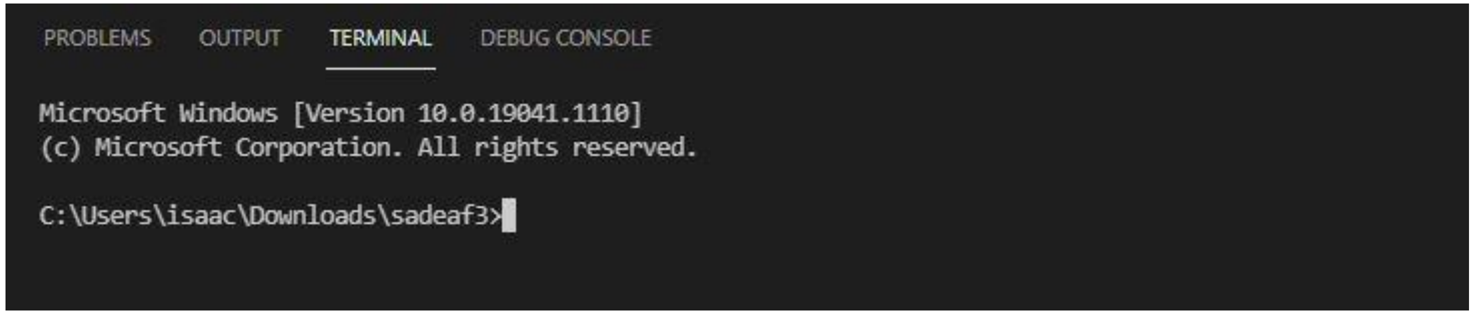
```
 "Hello, World" in C

dio.h>

here your code executes

Hello World\n");

will return:
gram is successful,
uccessful */
;
```

Click View -> Terminal.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\isaac\Downloads\sadeaf3>
```

The Terminal window appears at bottom of editor. Click on the top right and change to Command Prompt.



```
Microsoft Windows [Version 10.0.19041.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\isaac\Downloads\sadeaf3>
```

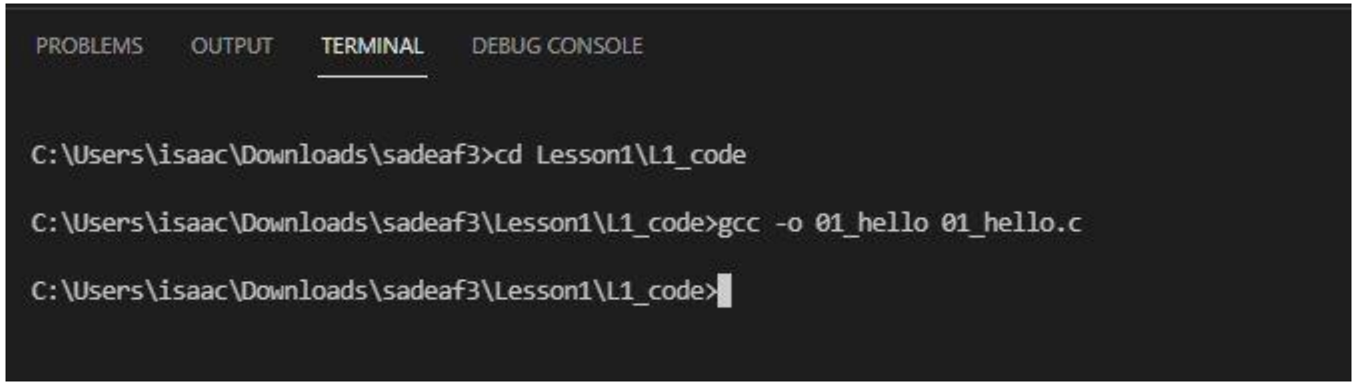
You are now using Command Prompt in the Visual Studio Code terminal.
By default, you will be in the directory where you opened your project.

Running our first program

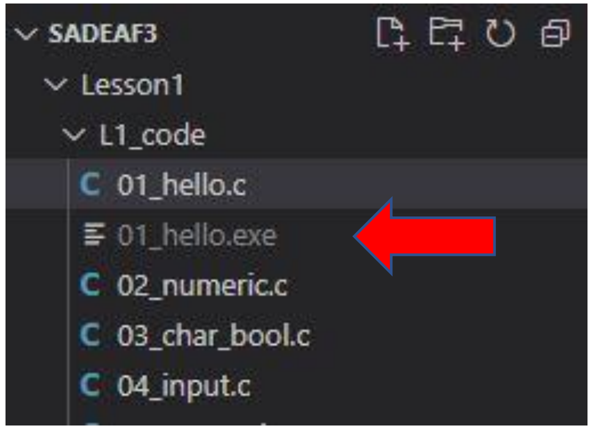
- We now have to run through steps 3-5 of writing a program:
    - Write the code
    - Compile code
    - Run the program
- We have already written the code.
- Now let’s compile our code. We will use a program called gcc to compile our code.

Using gcc to compile code

- We need to first change directories to the folder L1_code where our code resides.
- Type in the following and press Enter. This will compile our code and create a binary file called 01_hello.exe. This is the file that contains the machine language!

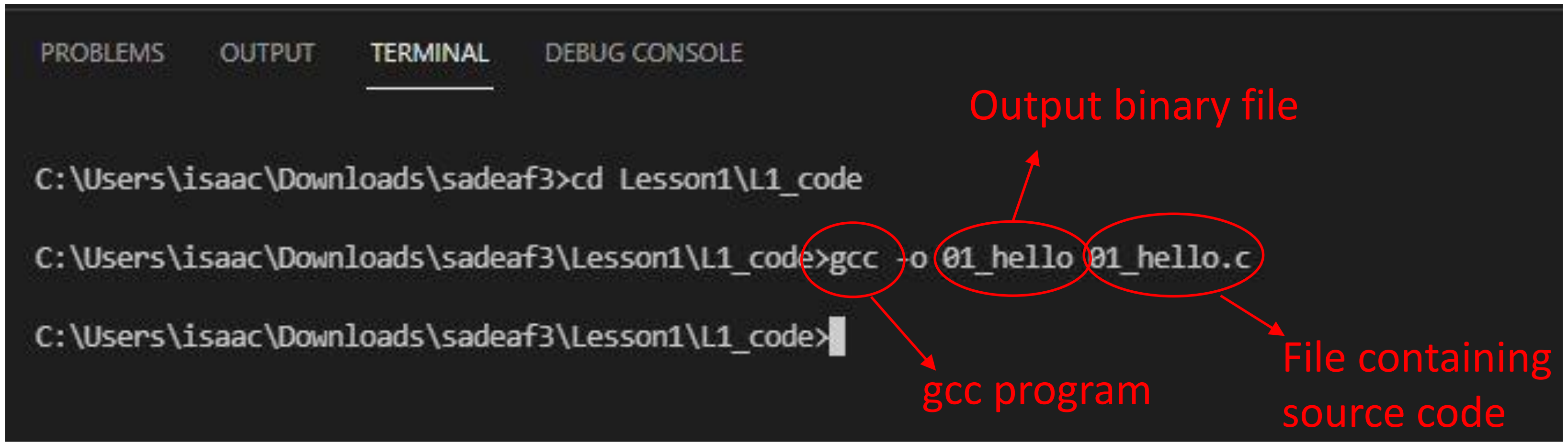


```
C:\Users\isaac\Downloads\sadeaf3>cd Lesson1\L1_code

C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code>gcc -o 01_hello 01_hello.c

C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code>
```



Using gcc to compile code

- Let’s take a closer look at what we typed into the command prompt

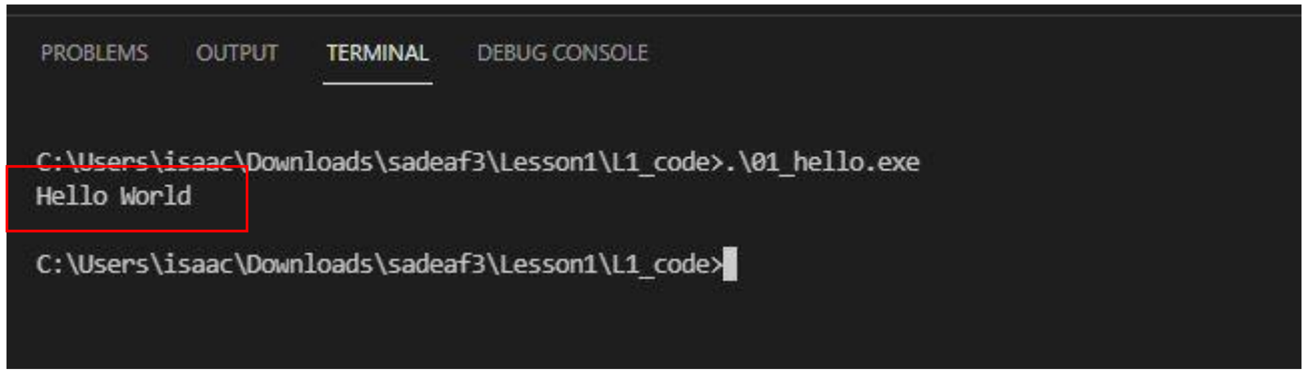


```
C:\Users\isaac\Downloads\sadeaf3>cd Lesson1\L1_code

C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code>gcc -o 01_hello 01_hello.c

C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code>
```

Hello World Output

- Now we need to run the program. Type the following to run the program, and we see our “Hello World” output.

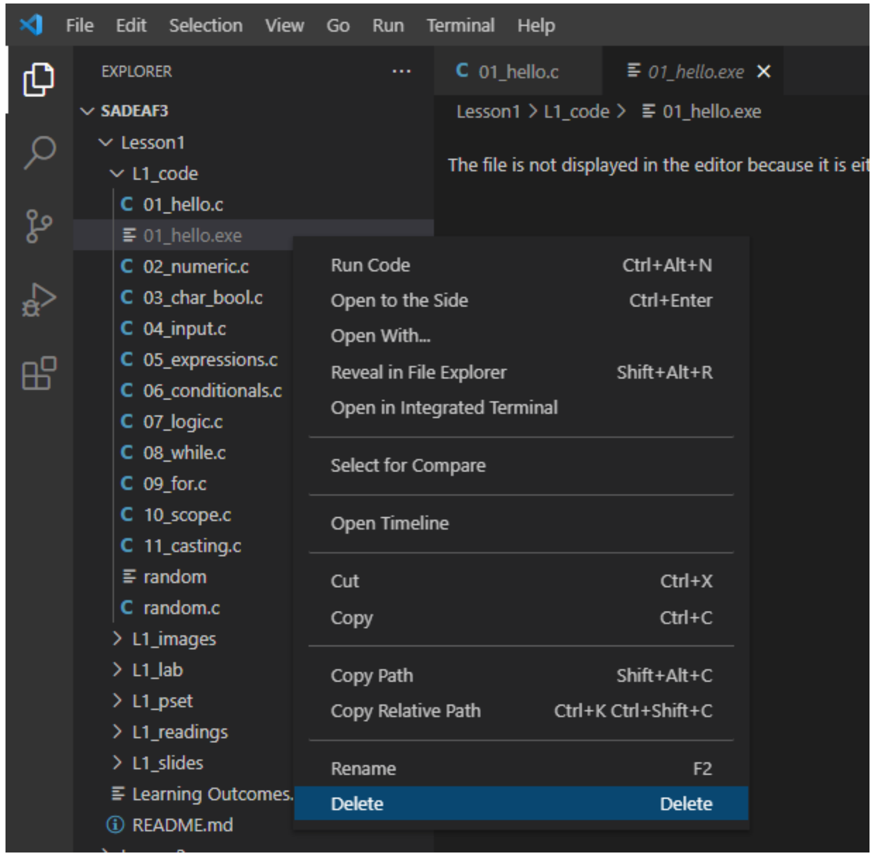
Output



```
C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code>.\01_hello.exe
Hello World

C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code>
```

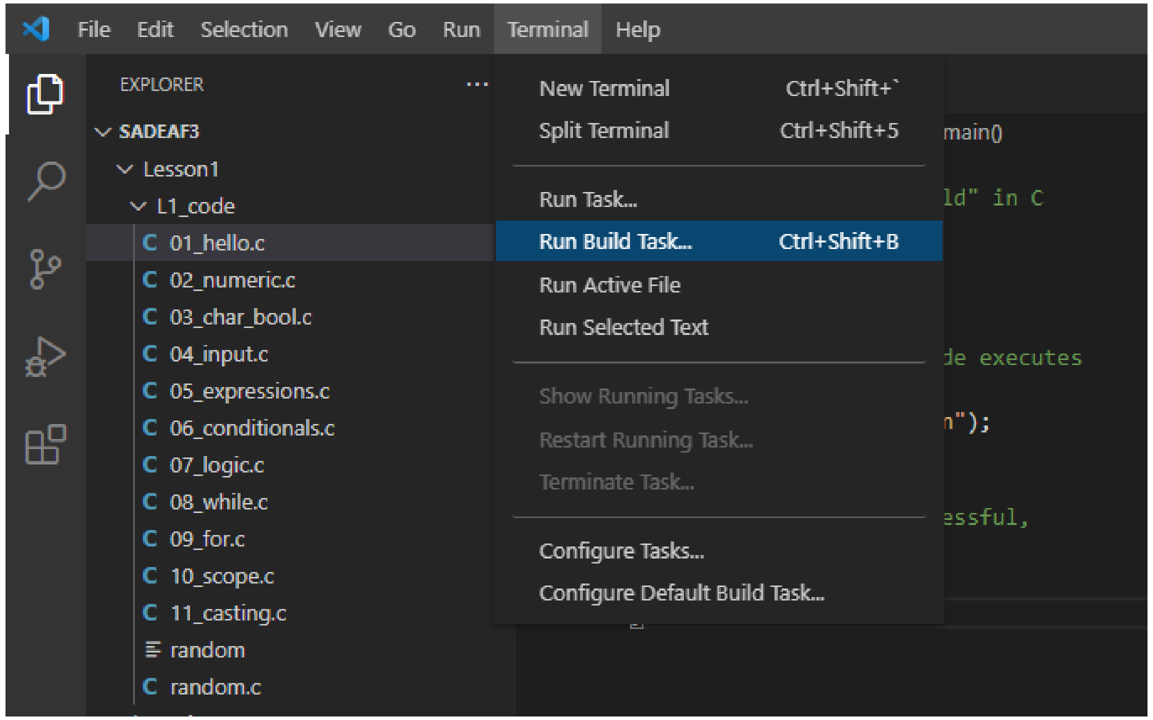
Using Run Build Task to compile code

- Now let’s look at a shortcut in Visual Studio Code to compile your code.
- Let’s first delete our binary file so we can see that the shortcut actually works.



The file is not displayed in the editor because it is ei

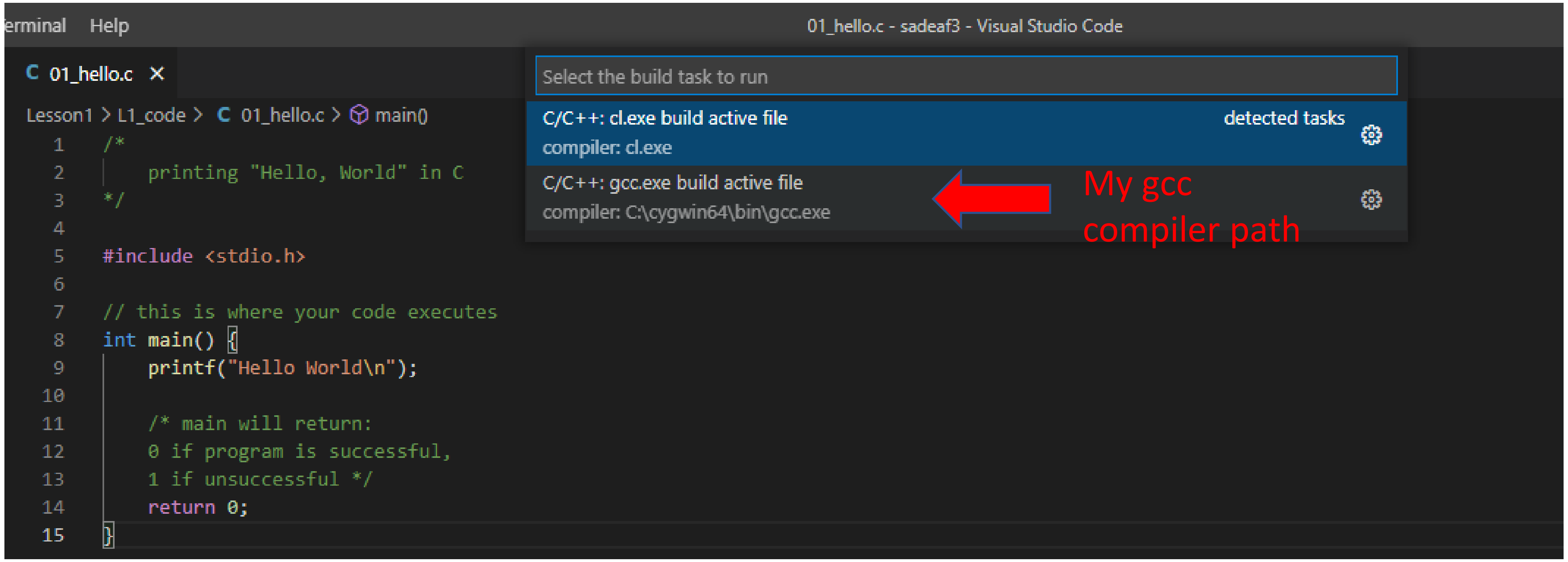
Using Run Build Task to compile code



```
main()
ld" in C
de executes
n");
essful,
```

- Next, with our file 01_hello.c open, click on Terminal -> Run Build Task

Using Run Build Task to compile code

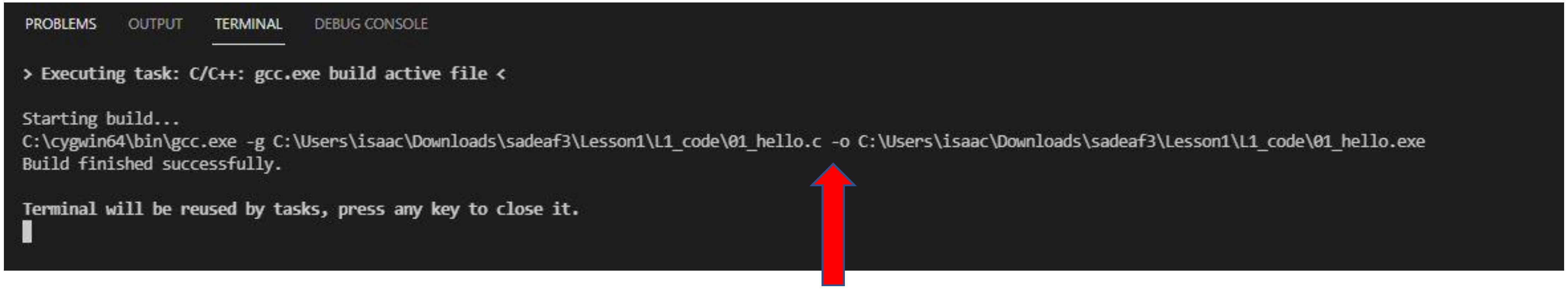

erminal
Help
01_hello.c - sadeaf3 - Visual Studio Code
01_hello.c
Lesson1 > L1_code > 01_hello.c > main()
Select the build task to run
detected tasks
My gcc compiler path

C/C++: cl.exe build active file
compiler: cl.exe

C/C++: gcc.exe build active file
compiler: C:\cygwin64\bin\gcc.exe

```
/*
    printing "Hello, World" in C
*/

#include <stdio.h>

// this is where your code executes
int main() {
    printf("Hello World\n");

    /* main will return:
    0 if program is successful,
    1 if unsuccessful */
    return 0;
}
```

A window should pop up asking you to choose your compiler. Choose your gcc compiler path



```
> Executing task: C/C++: gcc.exe build active file <

Starting build...
C:\cygwin64\bin\gcc.exe -g C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code\01_hello.c -o C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code\01_hello.exe
Build finished successfully.

Terminal will be reused by tasks, press any key to close it.
```

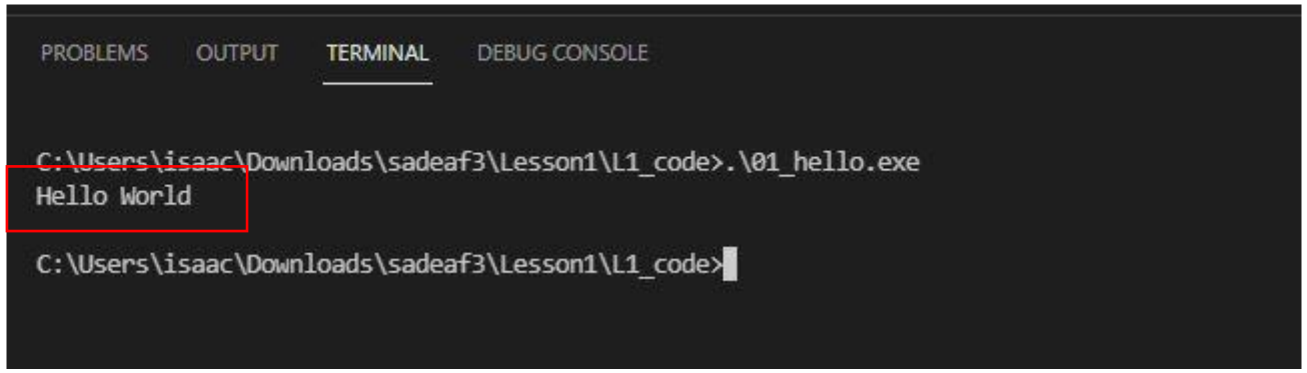
gcc compiling your code

Your code will now be compiled by gcc. A binary file called 01_hello.exe is once again created. Press Enter to go back to Command Prompt.

Hello World Output

- Now run the program again. Type the following to run the program, and we see our “Hello World” output.

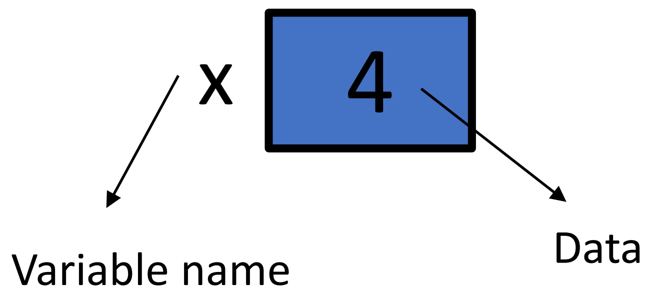
Output



```
C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code>.\01_hello.exe
Hello World

C:\Users\isaac\Downloads\sadeaf3\Lesson1\L1_code>
```

Variables

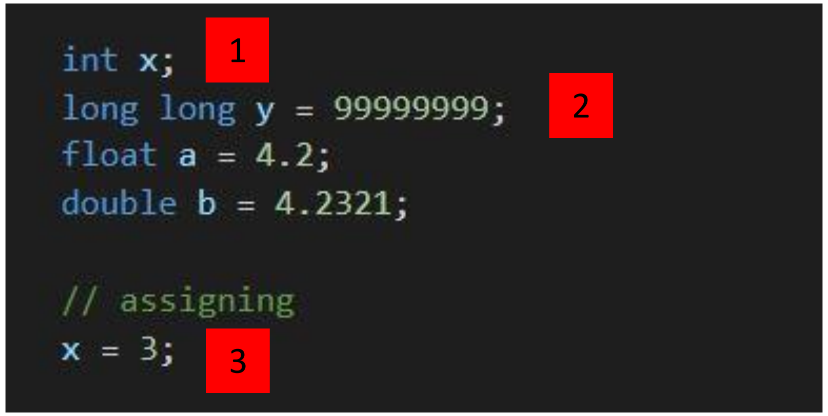
- To allow our programs to do more powerful things, they need to store and manipulate data. Variables are used to store our data types and save them for further use in the program. They are essentially names that we can use to access a specific data. You can also think of them as boxes in your computer's memory that hold the data.
- Variables of a certain data type can only hold data of that type



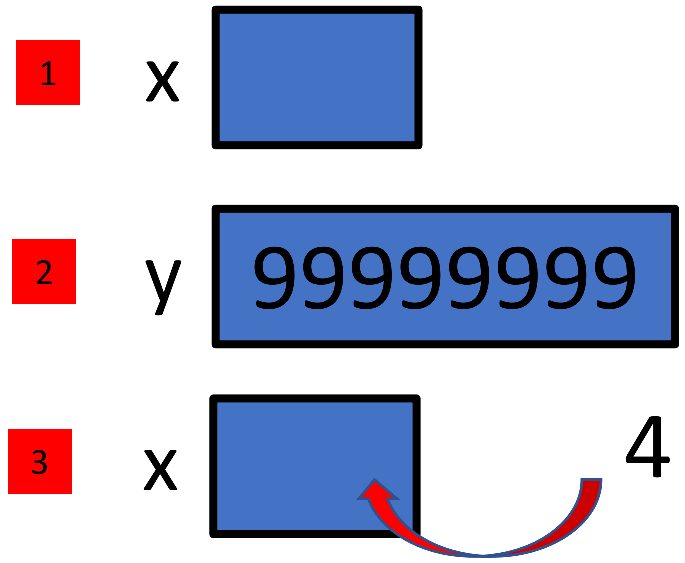
Naming variables

- There are rules to naming variables:
  - Variable names should only contain letters, numbers, or underscores
  - Variable names must begin with a letter
  - You can't have two variables in the same program with the same name
  - Valid variable names:
    - myData
    - pay94,
    - age_limit
    - Amount
    - Qt1yIncome
  - Invalid variable names:
    - 94Pay
    - my Age
    - lastname, firstname

Declare, initialize, assign variables



```
int x;                        1
long long y = 99999999;           2
float a = 4.2;
double b = 4.2321;

// assigning
x = 3;                        3
```



1 Declare/define variables – state the variable's type and name. This lets your computer know to allocate memory space for your variable

2 Initialize variables – sets an initial value for the variable when you declare it

3 Assign a value to a variable. The variable must have already been declared. Assigning a value uses the assignment operator (=) to put a value into the variable

Data types

- Examples of data types:
  - numbers, decimals, characters, strings etc.

Data types – numeric data types

- Integers
  - Whole numbers (can be signed or unsigned)
  - Range of signed integers: -2,147,483,648 to 2,147,483,647
  - Range of unsigned integers: 0 to 4,294,967,295
- Long long
  - Whole numbers like integers, but with bigger range
  - Range of signed long long: -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
  - Range of unsigned long long: 0 to 18,446,744,073,709,551,615
- Floats
  - Decimals (signed), range indicates the precision
  - Range: 3.4×10^{-38} to 3.4×10^{38}
- Doubles
  - Decimals (signed), higher precision than float
  - Range: 1.7×10^{-308} to 1.7×10^{308}

References:
https://docs.microsoft.com/en-us/cpp/cpp/data-type-ranges?view=msvc-160
https://docs.microsoft.com/en-us/cpp/c-language/storage-of-basic-types?view=msvc-160

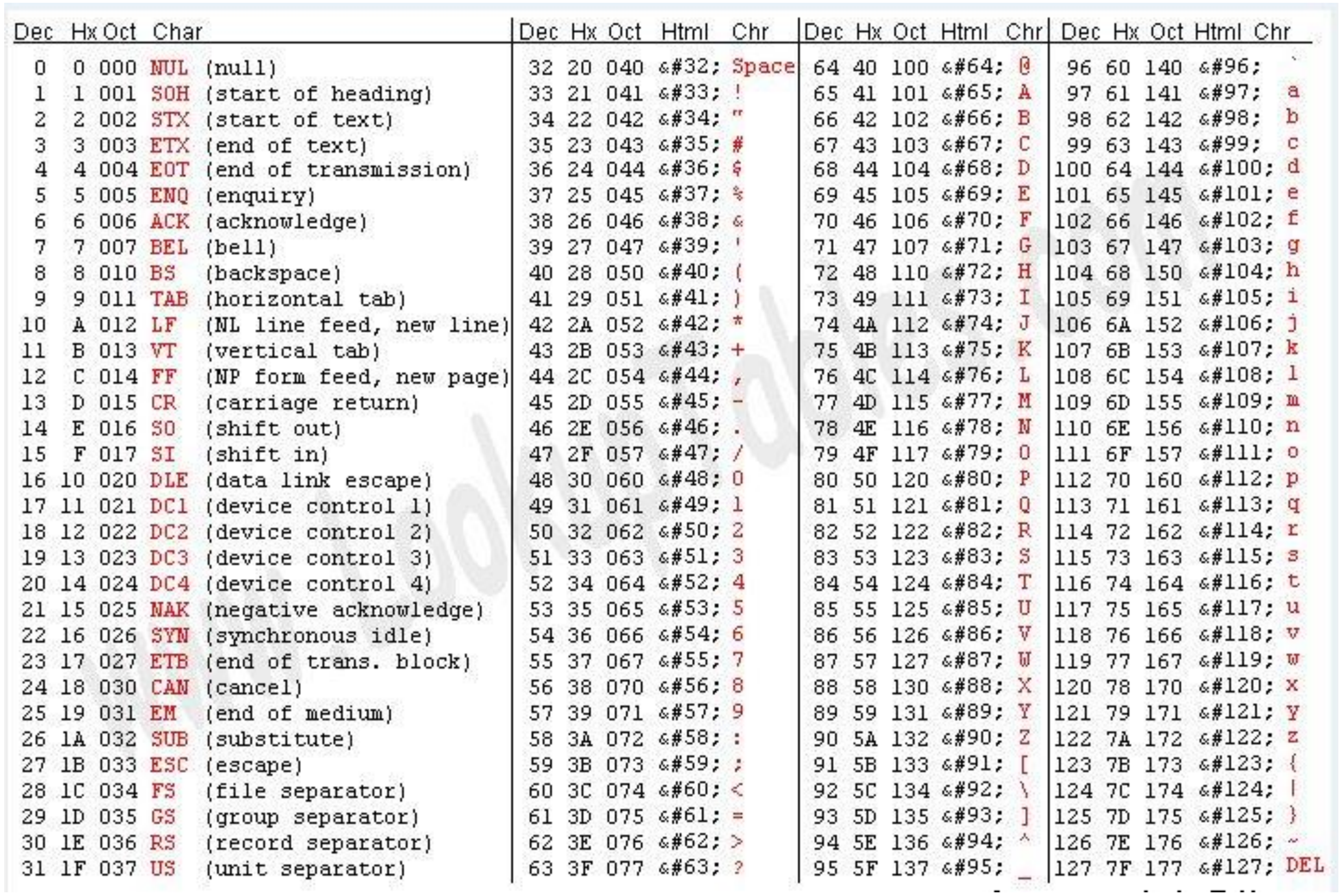
Data types – numeric data types

Data type	Examples
int	• 1 • -10,000 • 2,147,483,000
long long	• 1 • -10,000 • -36,854,775,808
float	• 1.00 • -3.1415 • 3762891.56743
double	• 1.00 • 0.0000001423 • 1.4×10^{-308}

Data types – char data type

- A character is any single character your computer can represent.
- Since computers can only understand numbers, an ASCII code is used as the numerical representation of a character. The ASCII table contains the mappings between ASCII codes and the characters they represent
- Your computer can understand 256 different characters, some printable, some non-printable.
- To specify printable characters, enclose them in single quotes - ‘’

Data types – char data type



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: https://www.asciitable.com/

Data types – char data type

- Some examples of characters:
  - ‘a’, ‘B’, ‘ ‘, ‘!’, ‘1’, ‘9’, ‘\n’
- Things to note:
  - ‘ ‘ indicates a space character
  - Characters such as ‘1’ and ‘9’ are characters and not integers
  - ‘\n’ is called the newline character

Data types – bool data type

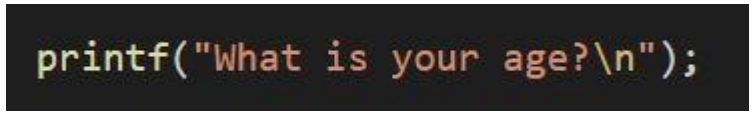
- The Boolean data type only contains 2 possible values – true or false.
- Booleans are also represented by numbers. 1 represents true, and 0 represents false
- Booleans are often used in conditional expressions, which we will learn later
- Examples of Booleans:
  - true, false

Any questions?

Input and output

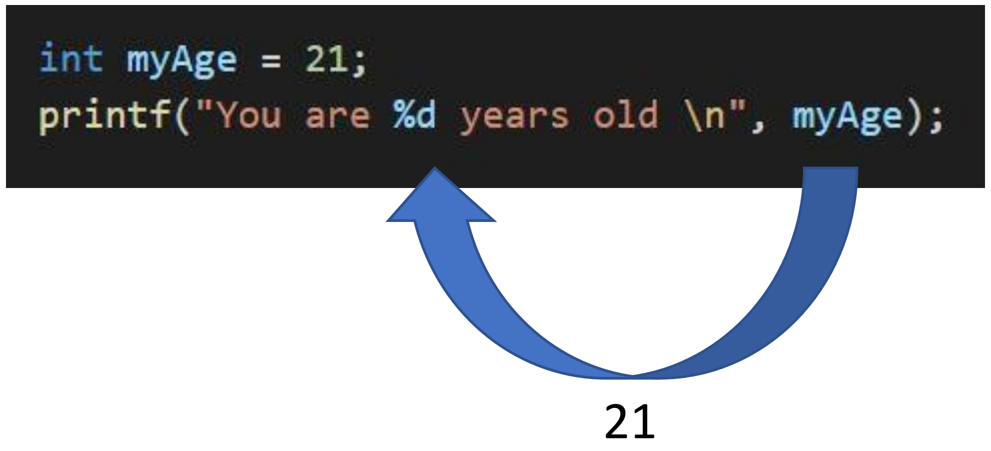
- For your program to interact with users, it needs to be able to read input and print output.
- `printf()` function prints formatted data onto the output screen
- `scanf()` function reads formatted data from standard input (stdin), which is usually the keyboard
- These 2 functions are built-in C functions that are part of the `stdio.h` library, the standard input/output library. A library is just a file containing code which other programmers can use. We call such files header files. To use the library, you need to include the header file in your program

Input and output – printf



```c
printf("What is your age?\n");
```

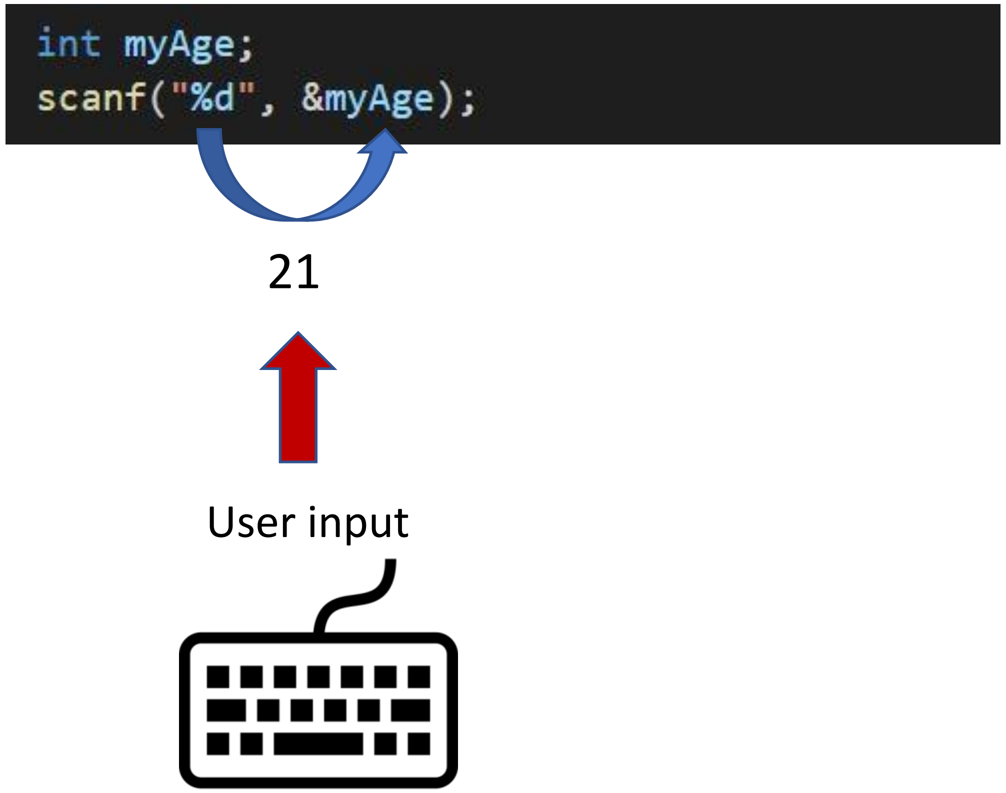
- The simplest use of the function printf() – takes in a C string as its only argument.



```c
int myAge = 21;
printf("You are %d years old \n", myAge);
```

- More complex use of the function printf() with more arguments – the C string can include format specifiers (sequences of characters beginning with %). The additional arguments are formatted and inserted into the string replacing their respective specifiers

Input and output – scanf



```c
int myAge;
scanf("%d", &myAge);
```

- Data is read from standard input according to the format specified and stored in the variables pointed to by the additional arguments

Input and output – format specifiers

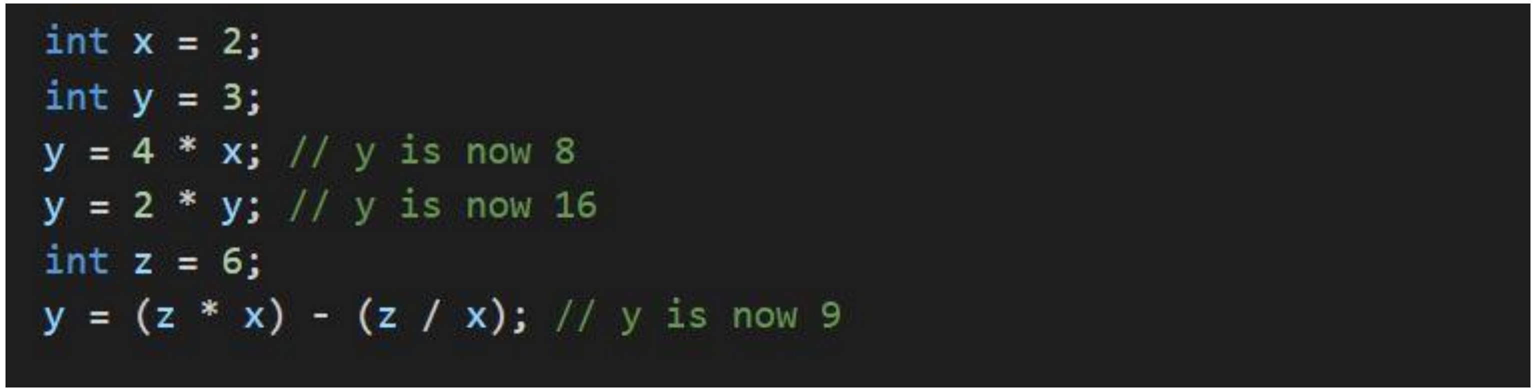
- Format specifiers lets your program know what is the data type of the data you want to print, so your program can print it correctly

Data type	Format specifier
int	%d
long long	%lld
float	%f
double	%lf
char	%c

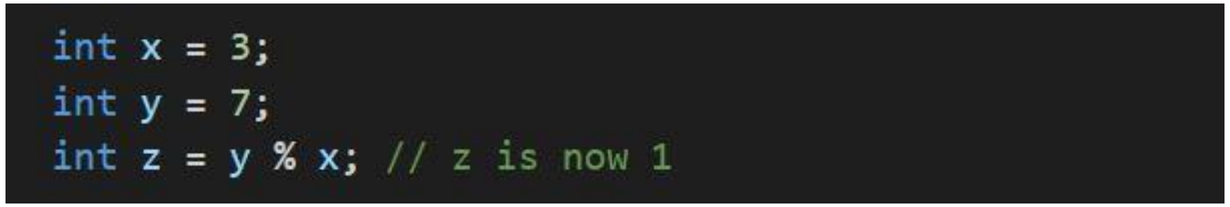
Arithmetic expressions

- One of the most fundamental things a computer needs to do is arithmetic!
- Operators (symbols used to perform a calculation or other task):
  - + (addition)
  - - (subtraction)
  - * (multiplication)
  - / (division)
  - % (modulus – calculates the remainder after a division)
  - = (assignment operator – assigns a value to a variable)
- Operands:
  - Variables or values on either side of the operators, which the operators work on
- Expressions:
  - A combination of operands and operators that results in a numeric value

Arithmetic expressions - examples

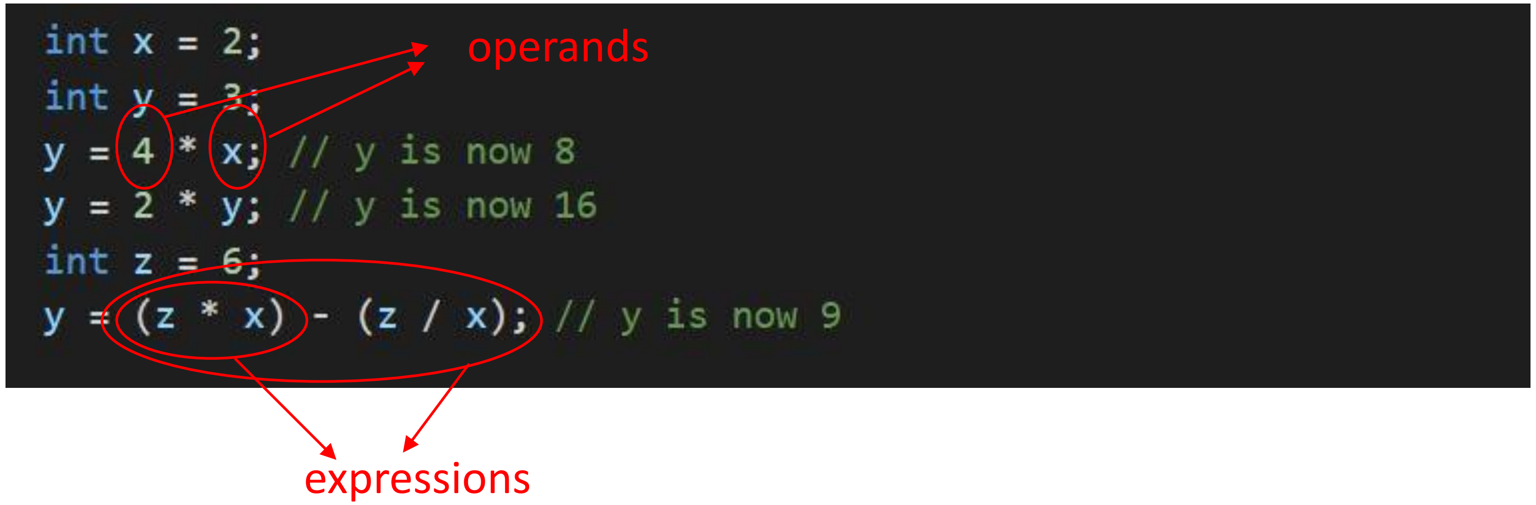


```
int x = 2;
int y = 3;
y = 4 * x; // y is now 8
y = 2 * y; // y is now 16
int z = 6;
y = (z * x) - (z / x); // y is now 9
```

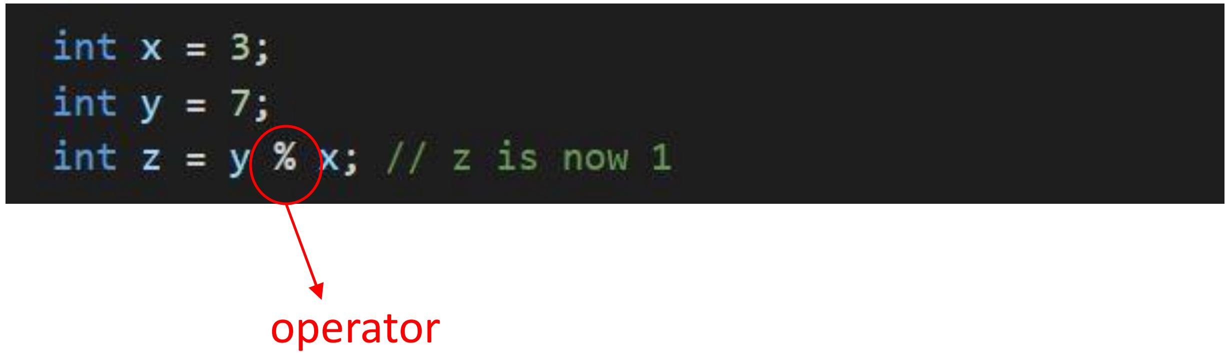


```
int x = 3;
int y = 7;
int z = y % x; // z is now 1
```

Arithmetic expressions - examples



```
int x = 2;
int y = 3;
y = 4 * x; // y is now 8
y = 2 * y; // y is now 16
int z = 6;
y = (z * x) - (z / x); // y is now 9
```



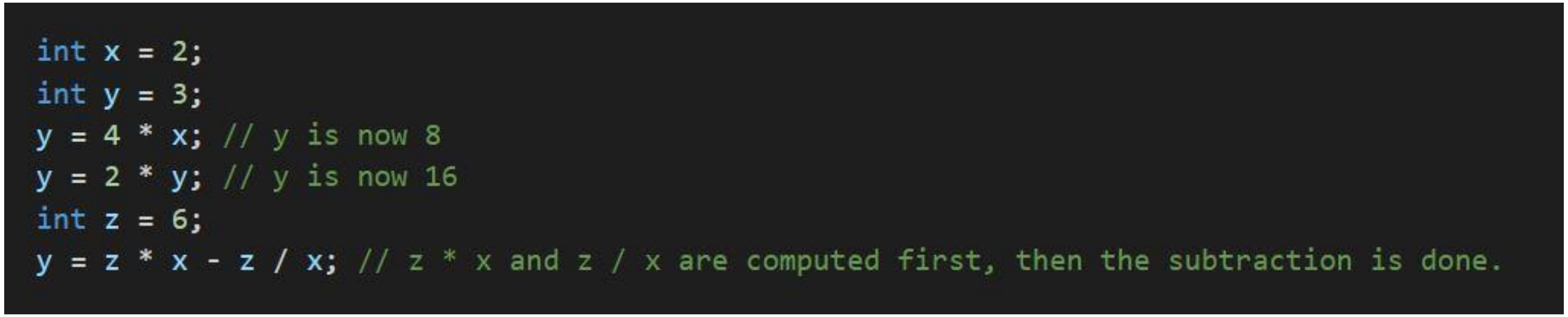
```
int x = 3;
int y = 7;
int z = y % x; // z is now 1
```

Arithmetic expressions – operator precedence

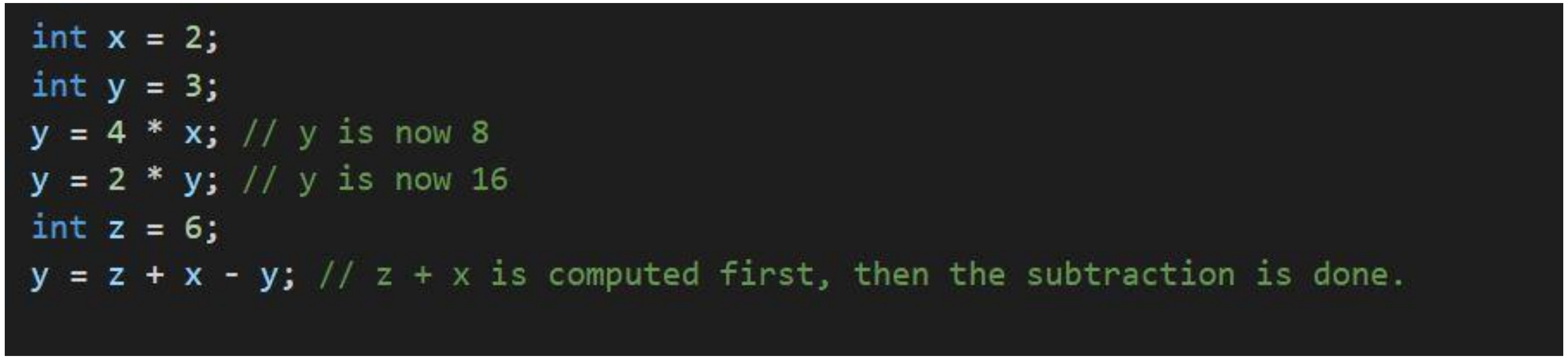
- Operator precedence – The order of precedence determines which operators act upon a value first.
- For operators with equal precedence, their associativity determines which operators act first (usually left to right)
- Expressions in opening and closing parentheses will be computed first

Order of precedence	Operator
Higher	* / %
Lower	+ -

Arithmetic expressions – operator precedence

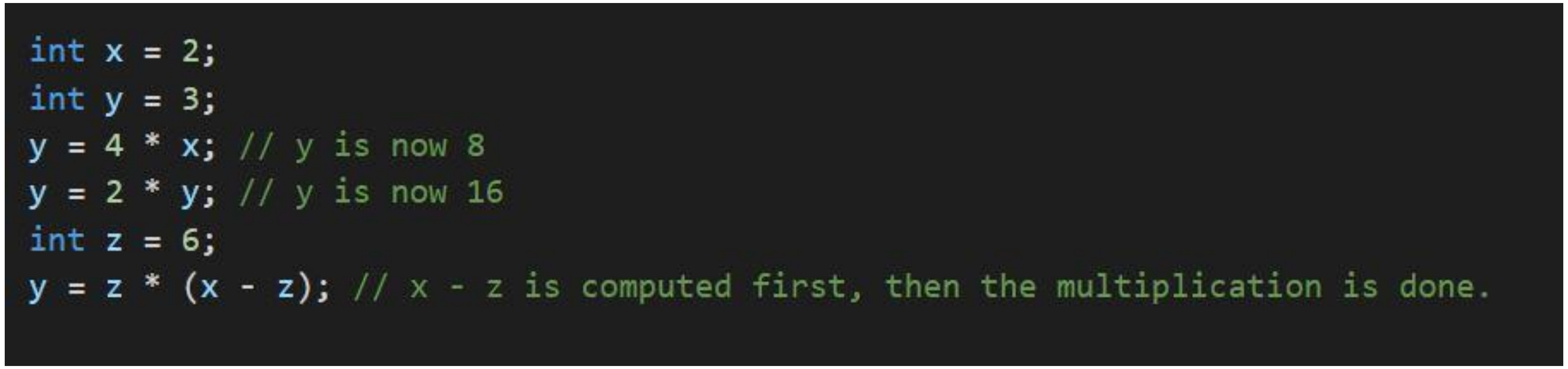


```
int x = 2;
int y = 3;
y = 4 * x; // y is now 8
y = 2 * y; // y is now 16
int z = 6;
y = z * x - z / x; // z * x and z / x are computed first, then the subtraction is done.
```



```
int x = 2;
int y = 3;
y = 4 * x; // y is now 8
y = 2 * y; // y is now 16
int z = 6;
y = z + x - y; // z + x is computed first, then the subtraction is done.
```

Arithmetic expressions – operator precedence

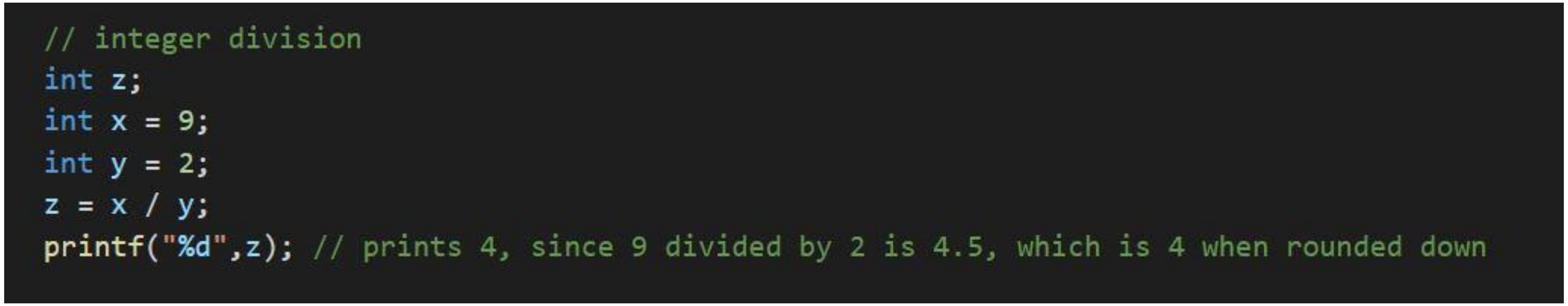


```
int x = 2;
int y = 3;
y = 4 * x; // y is now 8
y = 2 * y; // y is now 16
int z = 6;
y = z * (x - z); // x - z is computed first, then the multiplication is done.
```

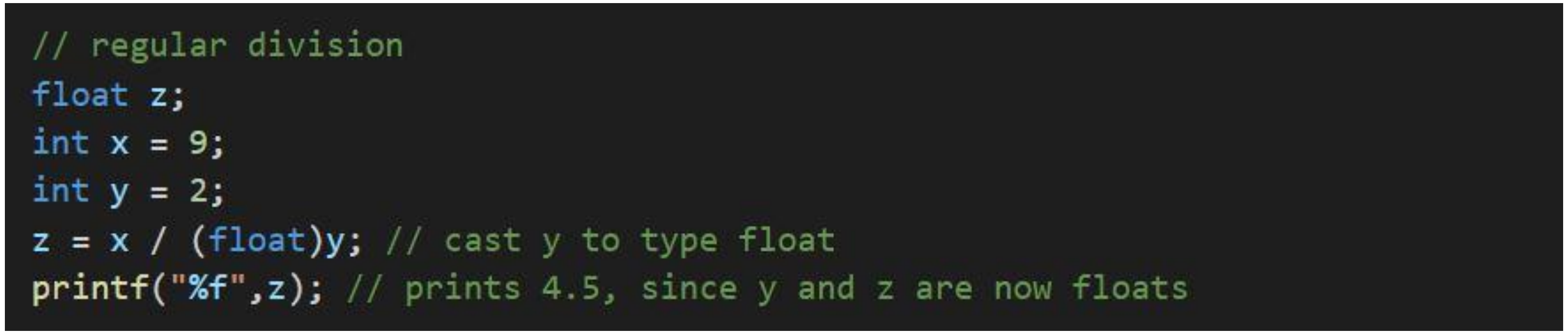
Arithmetic expressions - division

- Regular division vs integer division
- Regular division results in decimal numbers (floating-point types)
- Integer division results in whole numbers, and will be automatically rounded down if the first integer is not divisible by the second integer

Arithmetic expressions - division



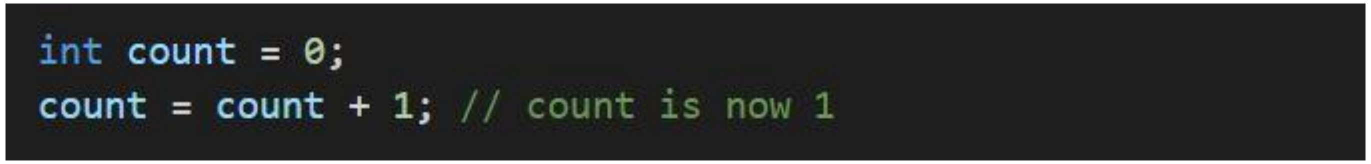
```c
// integer division
int z;
int x = 9;
int y = 2;
z = x / y;
printf("%d",z); // prints 4, since 9 divided by 2 is 4.5, which is 4 when rounded down
```



```c
// regular division
float z;
int x = 9;
int y = 2;
z = x / (float)y; // cast y to type float
printf("%f",z); // prints 4.5, since y and z are now floats
```

Arithmetic expressions – updating variables

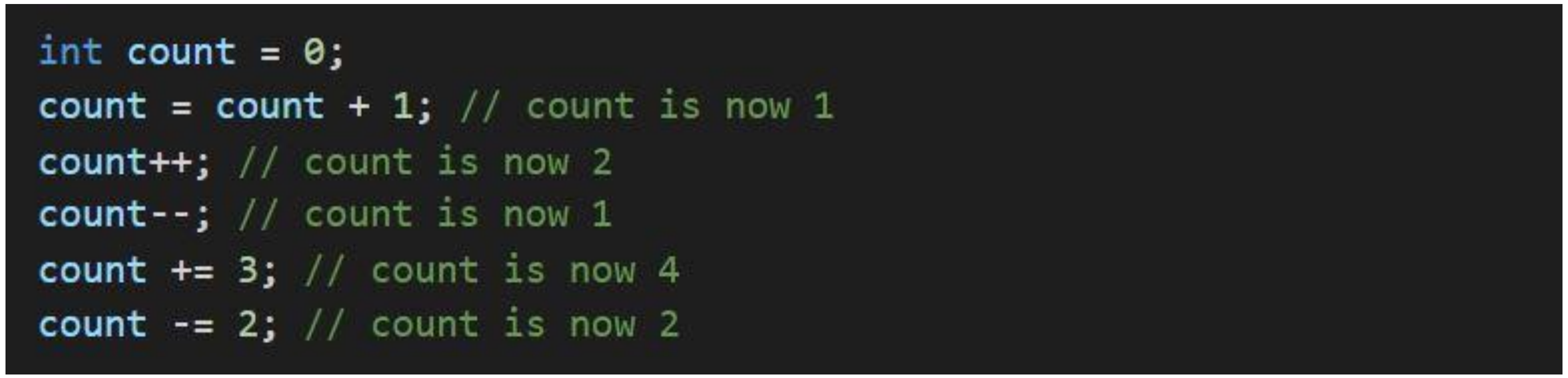
- To update the value of a variable, we can use an arithmetic expression to compute a new value, and store it back into the variable



```
int count = 0;
count = count + 1; // count is now 1
```

Arithmetic expressions – updating variables

- Postfix expressions are shortcuts to update the value of a variable. Examples include:
  - ++ (eg: x++, equivalent to x = x + 1)
  - -- (eg: x--, equivalent to x = x – 1)
  - += (eg: x += 1, equivalent to x = x + 1)
  - -= (eg: x -= 1, equivalent to x = x – 1)



```
int count = 0;
count = count + 1; // count is now 1
count++; // count is now 2
count--; // count is now 1
count += 3; // count is now 4
count -= 2; // count is now 2
```

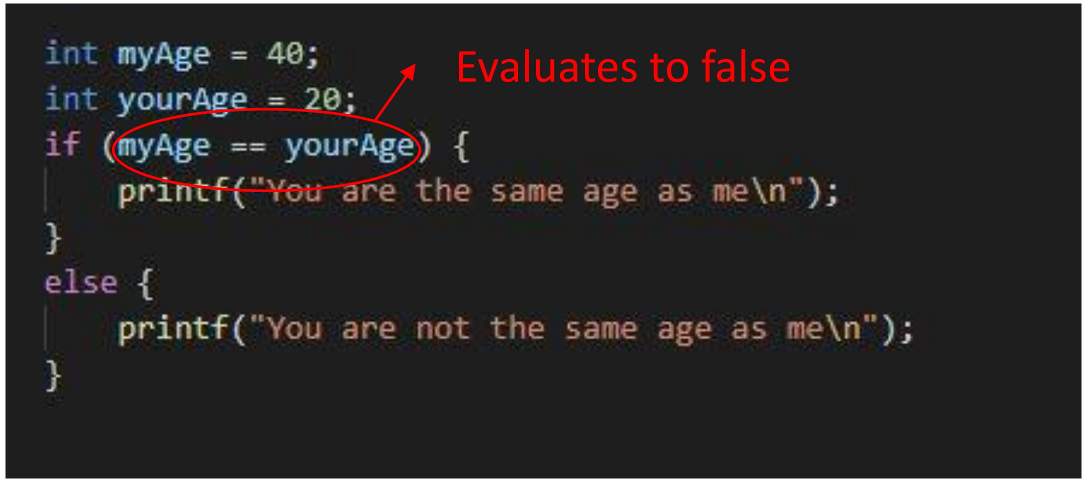
Conditional expressions

- Conditional statements help you to make a decision based on certain conditions.
- Examples of conditional statements:
  - If I earn more than $1000 this month, then we’ll go to Italy.
  - If the weather is more than 25 degrees Celsius, then wear shorts, else wear pants.
- In C, these conditions are specified by boolean expressions which evaluate to true or false

Conditional expressions - syntax

```
if (condition)
{ body }
else
{ body}
```

Note:
Curly braces are not required if there is only one statement in the if-else block's body. However, it is required if you have more than one statement. Hence, it is always good practice to put curly braces

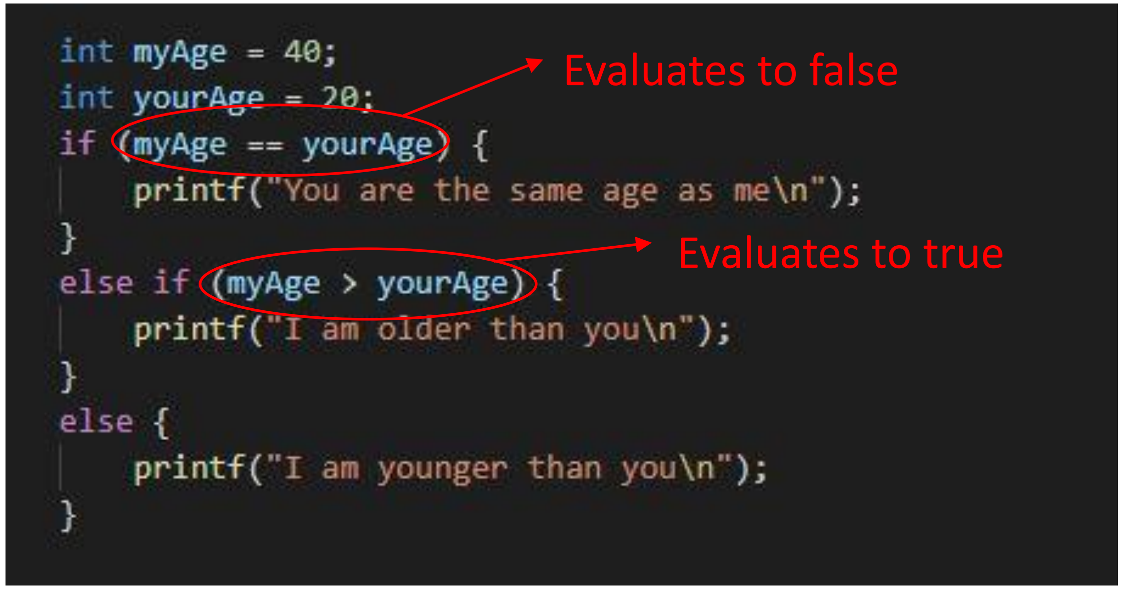


```
int myAge = 40;
int yourAge = 20;
if (myAge == yourAge) {
    printf("You are the same age as me\n");
}
else {
    printf("You are not the same age as me\n");
}
```

Output:
You are not the same age as me

Conditional expressions - syntax

```
if (condition)
{ body }
else if (condition)
{ body }
else if (condition)
{ body }
else
{ body }
```

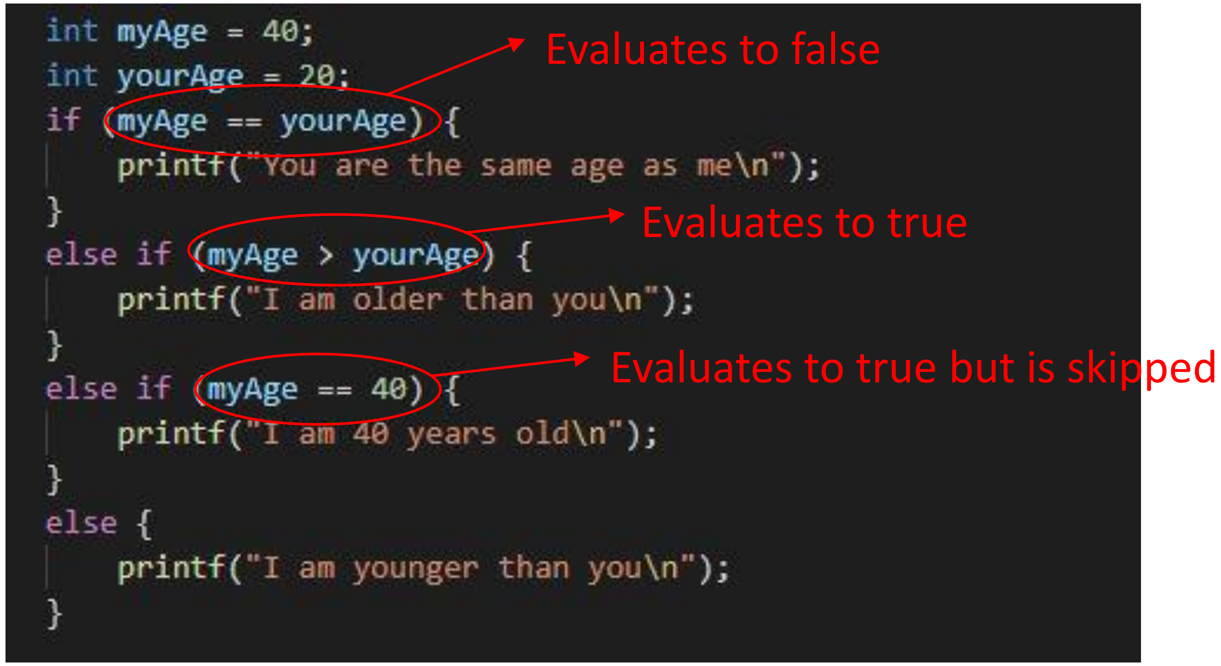


```
int myAge = 40;
int yourAge = 20;
if (myAge == yourAge) {
    printf("You are the same age as me\n");
}
else if (myAge > yourAge) {
    printf("I am older than you\n");
}
else {
    printf("I am younger than you\n");
}
```

Output:
I am older than you

Conditional expressions - syntax

```
if (condition)
{ body }
else if (condition)
{ body }
else if (condition)
{ body }
else
{ body }
```



```c
int myAge = 40;
int yourAge = 20;
if (myAge == yourAge) {
    printf("You are the same age as me\n");
}
else if (myAge > yourAge) {
    printf("I am older than you\n");
}
else if (myAge == 40) {
    printf("I am 40 years old\n");
}
else {
    printf("I am younger than you\n");
}
```

Note: If any condition evaluates to true, your program will run only the block of statements that correspond to that condition. It will skip the rest even if those conditions evaluate to true.

Output:
I am older than you

Comparison operators

- Comparison operators – operators that compare values and return true or false.

Comparison	Operator	Result
Less than	<	returns true if the value on the left is less than the value on the right, otherwise it returns false
Greater than	>	returns true if the value on the left is greater than the value on the right, otherwise it returns false.
Less than or equal to	<=	returns true if the value on the left is less than or equal to the value on the right, otherwise it returns false.
Greater than or equal to	>=	returns true if the value on the left is greater than or equal to the value on the right, otherwise it returns false.
Equal to	==	returns true if the value on the left is equal to the value on the right, otherwise it returns false.
Not equal to	!=	returns true if the value on the left is not equal to the value on the right, otherwise it returns false.

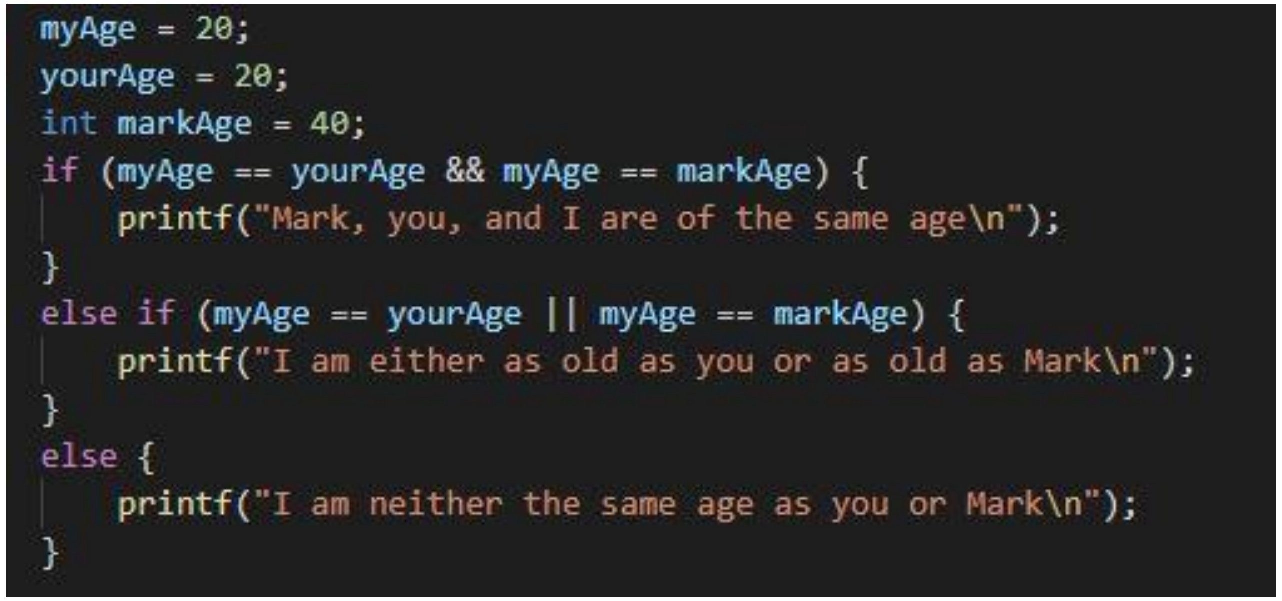
Logical (Boolean) operators

- Logical operators – operators that combine multiple boolean expressions or values and return true or false.
- Examples:
  - I weigh at least 100kg AND I am at least 2 meters tall
  - I have a degree in Computer Science OR I have a degree in Electrical Engineering
  - I am NOT (American OR British)
  - x is divisible by 2 AND x is divisible by 3

Logical (Boolean) operators

Logical operation	Operator	Result
AND	&&	Returns true if and only if the expressions on both sides of it are true. Otherwise, returns false
OR	||	Returns true if the expression on either side of it is true. Otherwise, returns false.
NOT	!	Reverses the logical value of its expression. If a condition is true, the NOT operator will make it false. If a condition is false, the NOT operator will make it true

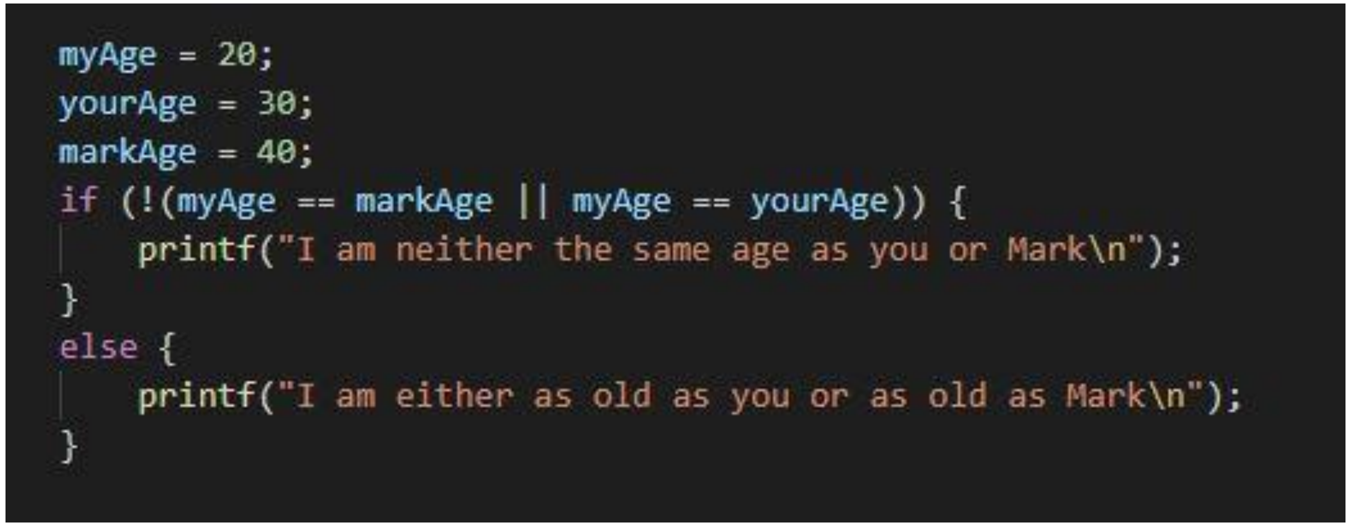
Logical (Boolean) operators



```c
myAge = 20;
yourAge = 20;
int markAge = 40;
if (myAge == yourAge && myAge == markAge) {
    printf("Mark, you, and I are of the same age\n");
}
else if (myAge == yourAge || myAge == markAge) {
    printf("I am either as old as you or as old as Mark\n");
}
else {
    printf("I am neither the same age as you or Mark\n");
}
```

Output:
I am either as old as you or
as old as Mark

Logical (Boolean) operators

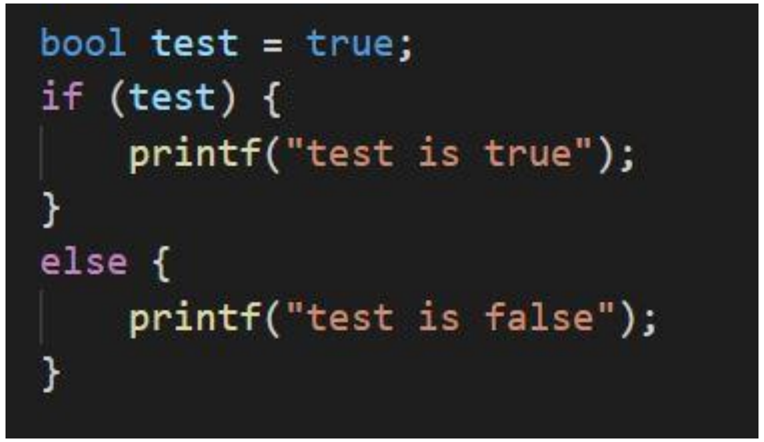


```
myAge = 20;
yourAge = 30;
markAge = 40;
if (!(myAge == markAge || myAge == yourAge)) {
    printf("I am neither the same age as you or Mark\n");
}
else {
    printf("I am either as old as you or as old as Mark\n");
}
```

Output:
I am neither the same age as you or Mark

Conditional expressions - syntax

- Boolean expressions evaluate to true or false, so you can just put a true or false value in the condition without making any comparison

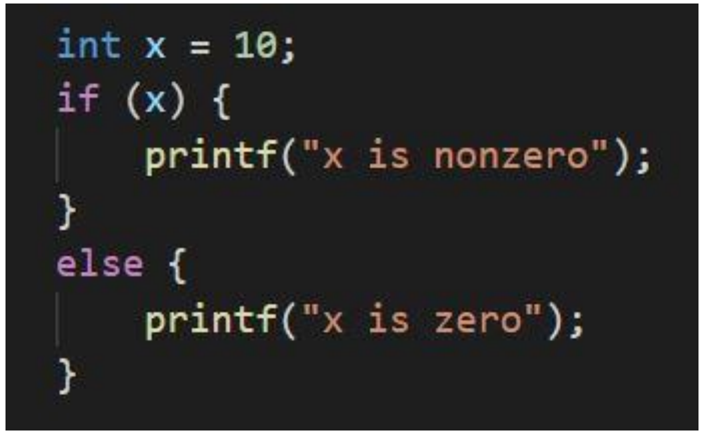


```
bool test = true;
if (test) {
    printf("test is true");
}
else {
    printf("test is false");
}
```

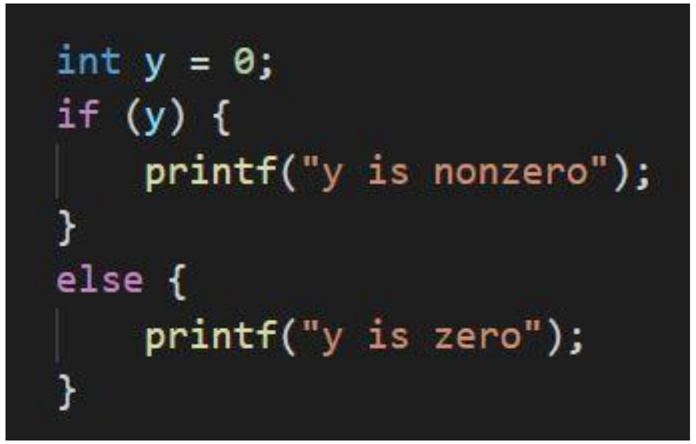
Output:
test is true

Conditional expressions - syntax

- When interpreting Boolean expressions, zero is interpreted as false, and anything non-zero is interpreted as true



```
int x = 10;
if (x) {
    printf("x is nonzero");
}
else {
    printf("x is zero");
}
```



```
int y = 0;
if (y) {
    printf("y is nonzero");
}
else {
    printf("y is zero");
}
```

Output:
x is nonzero

Output:
y is zero

Conditional expressions - opposites

Expression	Opposite
NOT (A > B)	A <= B
NOT (A < B)	A >= B
NOT (A <= B)	A > B
NOT (A >= B)	A < B
NOT (A == B)	A != B
NOT (A != B)	A == B

Switch statements

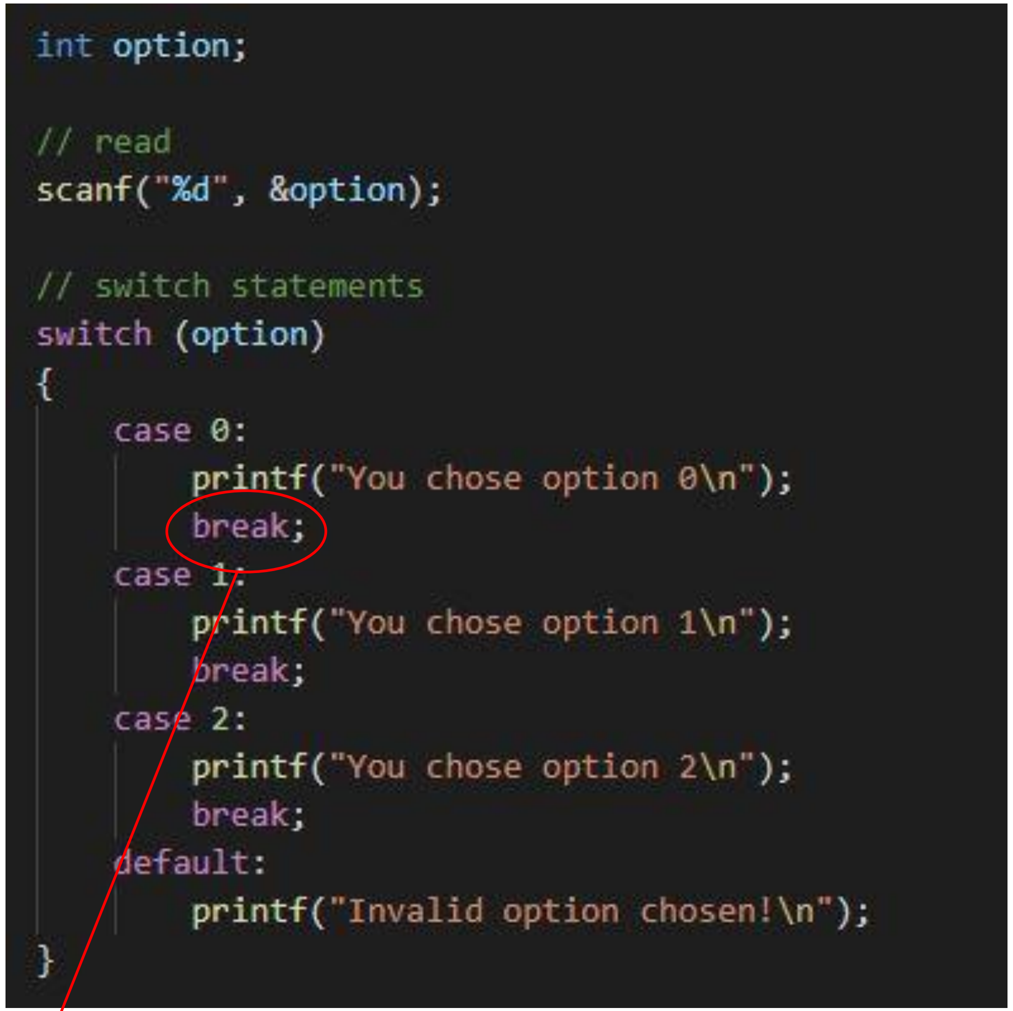
- The if-else statement is great for simple testing of data, like if your data has only 2 or 3 possible values. However if your data can take on many possible values, you will need many else-if blocks, and this can make your code messy.
- The switch statement is a better way of testing many different possible values.

Switch statements - syntax

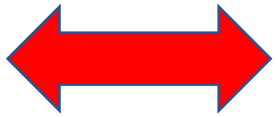
```
switch (variable)
{
  case value1:
  { body }
  case value2:
  { body }
  case value1:
  { body }
  case value2:
  { body }
  default:
  { body }
}
```

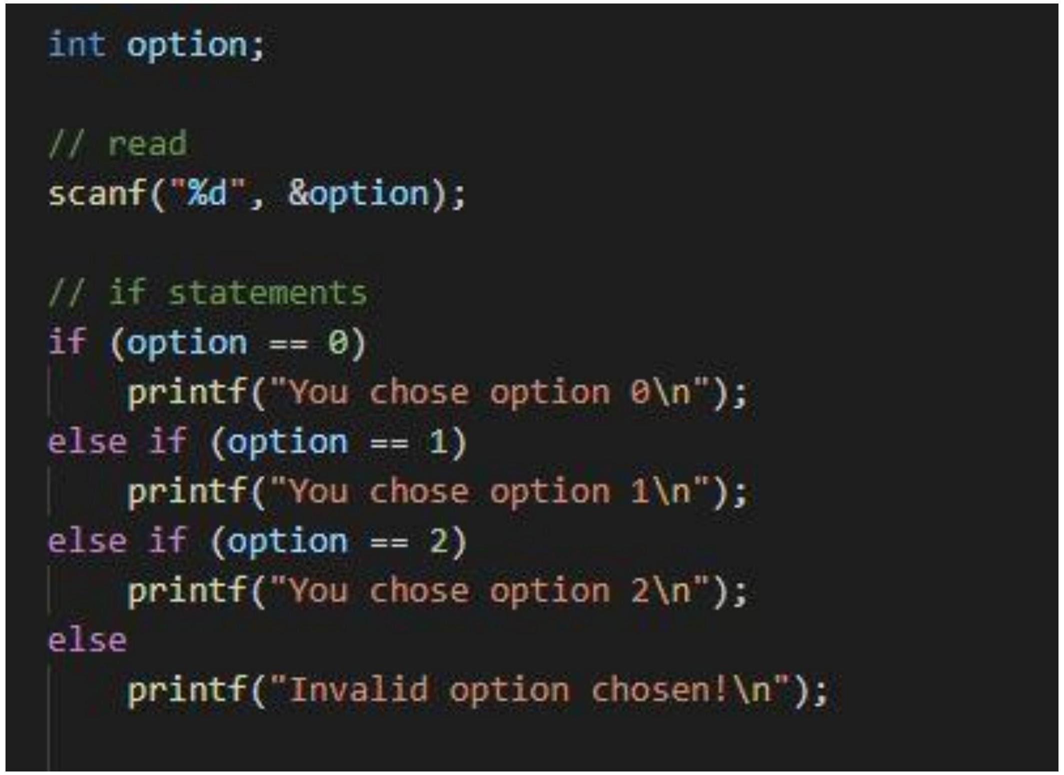
similar

```
if (condition)
{ body }
else if (condition)
{ body }
else if (condition)
{ body }
else if (condition)
{ body }
else
{ body }
```



```
int option;

// read
scanf("%d", &option);

// switch statements
switch (option)
{
    case 0:
        printf("You chose option 0\n");
        break;
    case 1:
        printf("You chose option 1\n");
        break;
    case 2:
        printf("You chose option 2\n");
        break;
    default:
        printf("Invalid option chosen!\n");
}
```





```
int option;

// read
scanf("%d", &option);

// if statements
if (option == 0)
    printf("You chose option 0\n");
else if (option == 1)
    printf("You chose option 1\n");
else if (option == 2)
    printf("You chose option 2\n");
else
    printf("Invalid option chosen!\n");
```

Note that this break statement is necessary for each switch case, or else your program will flow through to the next switch case

Loops

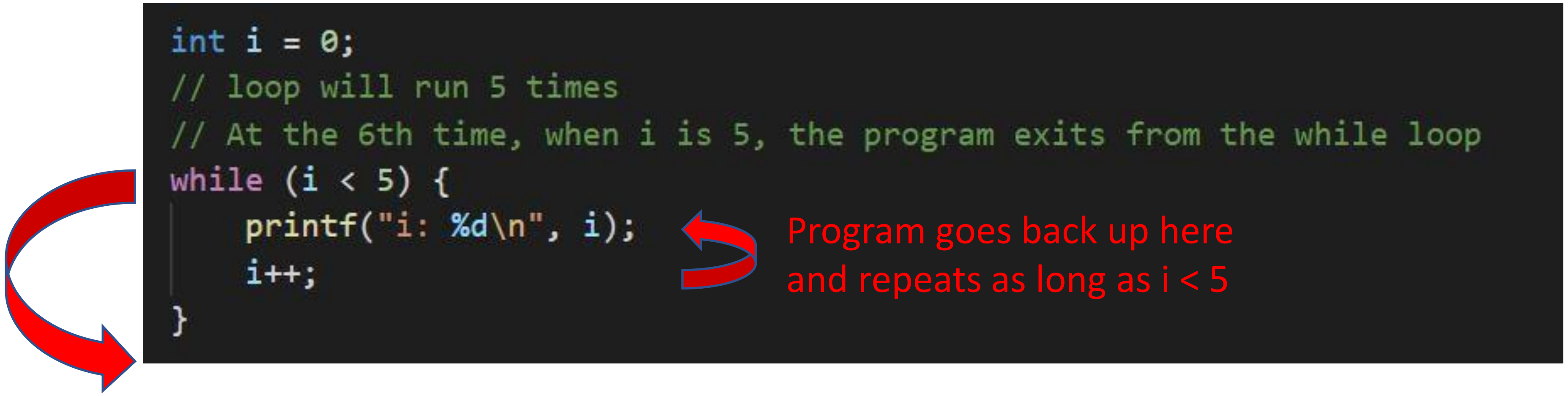
- A loop is a section of code that repeats a number of times. This is known as *iteration*. You don’t want a loop to repeat forever – that’s called an infinite loop. Loops should eventually come to a stop once they finish doing the job.
- 2 types of loops – while and for loop
- The block of code inside a loop is repeated as long as the looping condition is met

Loops – while loop

```
while (condition)
{ body }
```

- The block of code within the curly braces repeats as long as (while) the condition is true.
- The possible conditions for a while loop are exactly the same as conditions used for if-else statements

Loops – while loop



```
int i = 0;
// loop will run 5 times
// At the 6th time, when i is 5, the program exits from the while loop
while (i < 5) {
    printf("i: %d\n", i);
    i++;
}
```

When i is 5, program exits from while loop,
and continues with the rest of the code

Note:
You must always remember to update the variable inside the while loop’s condition, or else your loop will repeat forever!

Loops – do-while loop

- A do-while loop behaves almost exactly like a while loop, except that the body of the loop executes at least one time.

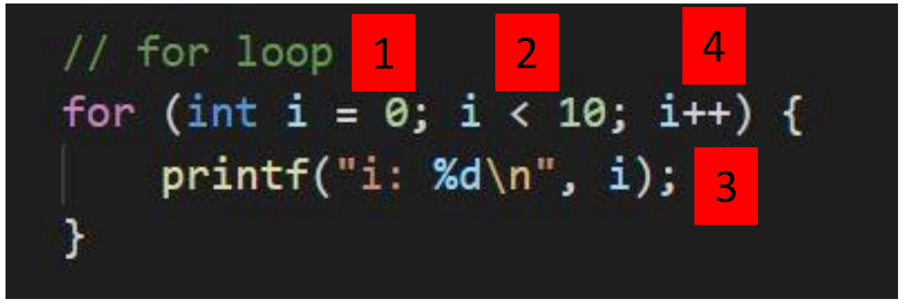
```
do
{ body }
while (condition)
```

Loops – for loop

```
for (declare and initialize variable; condition;
update variable)
{ body }
```

- A variable is first declared and initialized.
- The variable is tested in the condition
- The block of code within the curly braces repeats as long as the condition is true.
- After each repetition of the for loop, the variable is updated, and then tested again in the condition

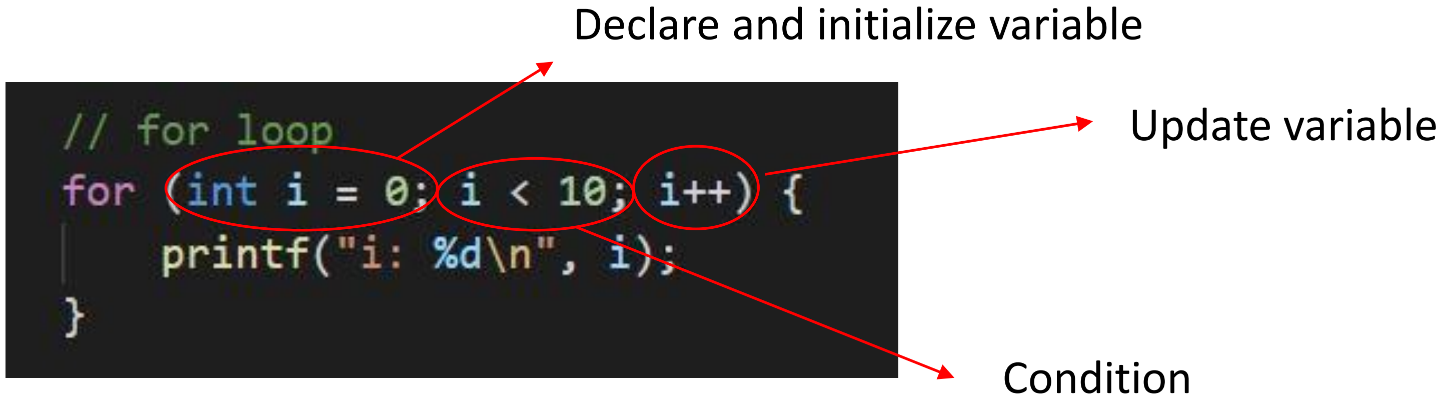
Loops – for loop



```
// for loop
for (int i = 0; i < 10; i++) {
    printf("i: %d\n", i);
}
```

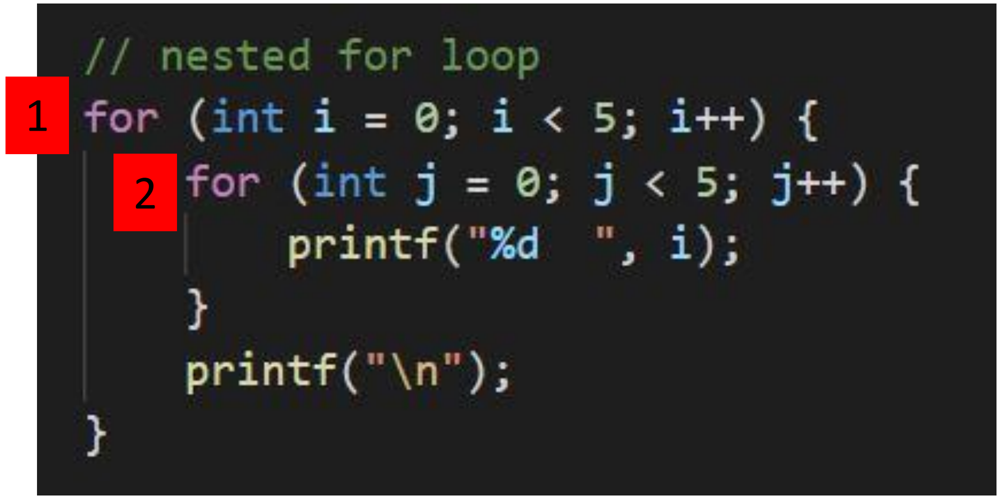
Sequence of operations:
1, 2, 3, 4, 2, 3, 4, 2, 3, 4, …

Loop will run 10 times.



```
// for loop
for (int i = 0; i < 10; i++) {
    printf("i: %d\n", i);
}
```

Loops – nested for loops



```
// nested for loop
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        printf("%d  ", i);
    }
    printf("\n");
}
```

For each repetition of loop 1 (outer loop), loop 2 (inner loop) repeats 5 times.

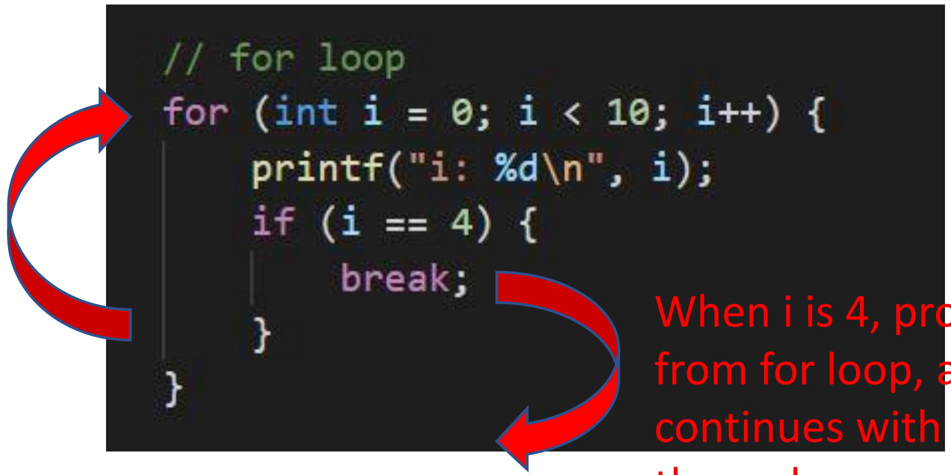
Output:

```
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
```

Loops – break and continue statements

- The break statement causes the program to exit the loop early
- The continue statement causes the program to immediately go to the next repetition of the loop
- These statements can be used in both for and while loops

Loops – break and continue statements



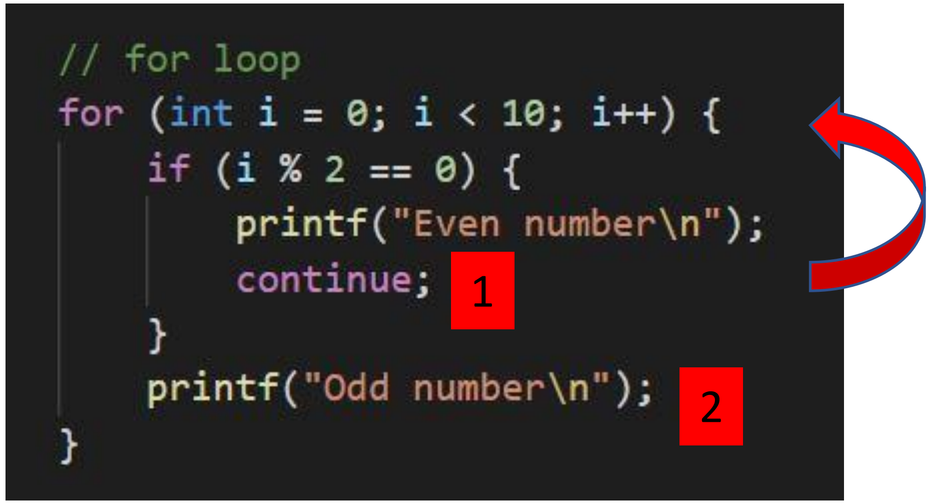
```c
// for loop
for (int i = 0; i < 10; i++) {
    printf("i: %d\n", i);
    if (i == 4) {
        break;
    }
}
```

Program goes back up here and repeats as long as i < 10

When i is 4, program exits from for loop, and continues with the rest of the code

Loop will run only 5 times instead of 10 times

Loops – break and continue statements



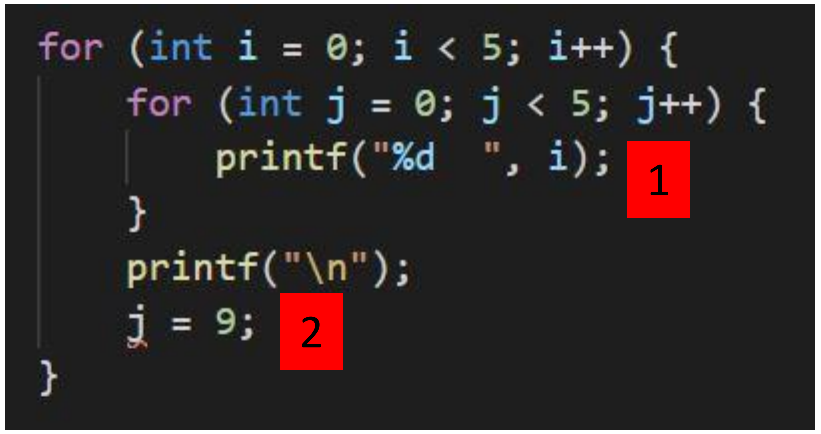
```
// for loop
for (int i = 0; i < 10; i++) {
    if (i % 2 == 0) {
        printf("Even number\n");
        continue;
    }
    printf("Odd number\n");
}
```

At 1 ,Program immediately goes to the next repetition of the loop and doesn't go to statement 2 when i is even

Variable scope

- A variable’s scope refers to the block of code in a program in which it is available in.
- When you create a variable within a block, it’s scope consists of the block it was created in, as well as any nested blocks within.
- Within a block, you can only access variables that have been initialized within that block and outer scoped blocks, but not vice versa.

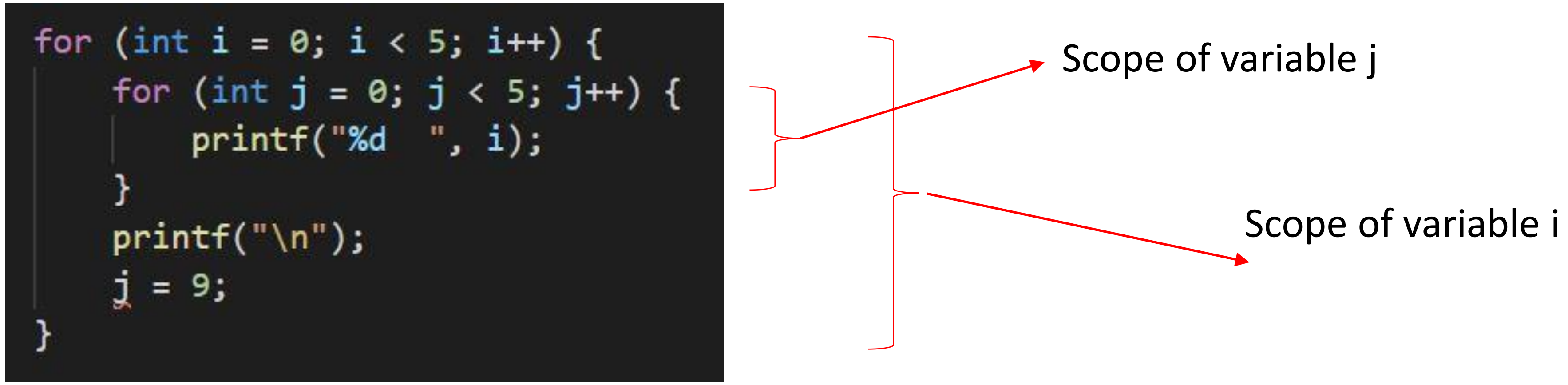
Variable scope



```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        printf("%d  ", i);
    }
    printf("\n");
    j = 9;
}
```

1 The inner loop can access the variable i because it was initialized in the outer loop. The scope of variable i includes both the outer and inner loops

2 The outer loop cannot access the variable j because it was initialized in the inner loop. The scope of variable j only includes the inner loop



```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 5; j++) {
        printf("%d  ", i);
    }
    printf("\n");
    j = 9;
}
```

Any questions?