CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 3.1: Bitwise operators, functions

Toronto Metropolitan University

Notice!

Obligatory copyright notice in the age of digital delivery and online classrooms:

The copyright to this original work is held by Alex Ufkes. Students registered in course CPS 188 can use this material for the purposes of this course but no other use is permitted, and there can be no sale or transfer or use of the work for any other purpose without explicit permission of Alex Ufkes.

© Alex Ufkes, 2023

Today

- User-defined functions
- Bitwise operators
- Bit masking

© Alex Ufkes, 2023

Function

A **named block of statements** that performs some operation that is intended to be performed more than once.

In other words, a piece of code we can reuse without having to duplicate the statements.

Functions make a program more **readable**, **simplify** coding, and allow for code **reuse** between programmers.

© Alex Ufkes, 2020, 2021

Functions

We've *used* many:

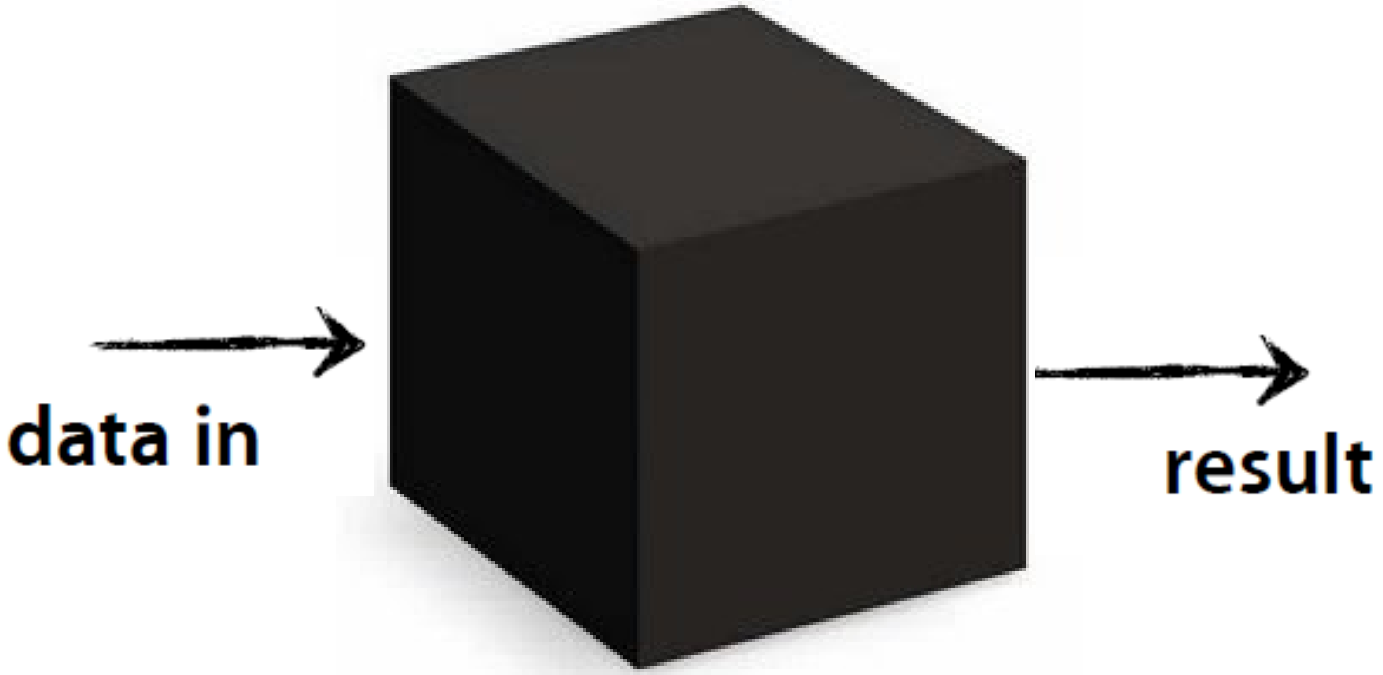
In **stdio.h**:
- `printf()`
- `scanf()`

In **`math.h`**:
- `sqrt()`, `pow()`
- `sin()`, `cos()`, `tan()`

- Each of these implements behavior provided to us by C.
- We are not responsible for writing code to compute square root. Someone else did that for us.

© Alex Ufkes, 2020, 2021

Functions as Black Boxes

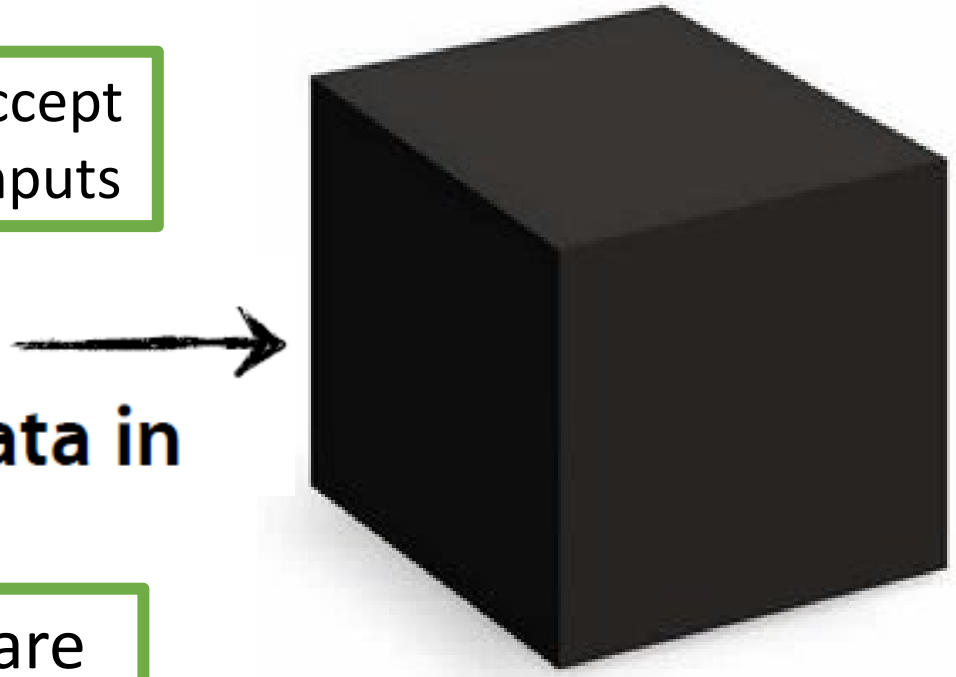


- We know what goes in, we know what comes out (ideally)
- We don’t care what goes on inside (for now)

© Alex Ufkes, 2020, 2021

Functions as Black Boxes

A function can accept any number of inputs



A function can return *at most* one result

These inputs are called ***arguments***

Any of the data types we've seen thus far can be returned

© Alex Ufkes, 2020, 2021

Functions

Input arguments & return values

`sqrt(15.8)`

`sqrt()` takes <u>one</u> input argument of type **`double`**

`sqrt()` returns one value of type **`double`**

© Alex Ufkes, 2020, 2021

Functions

Multiple input arguments:


```
x = scanf("%d", &num)
```

`scanf()` takes <u>two</u> input arguments here: placeholder string, variable address.

`scanf()` returns one value of type `int`

© Alex Ufkes, 2020, 2021

Functions

Can we have multiple results?

NO.

A function returns a **single** result *(technically)*.

© Alex Ufkes, 2020, 2021

Functions

No arguments? No return values?

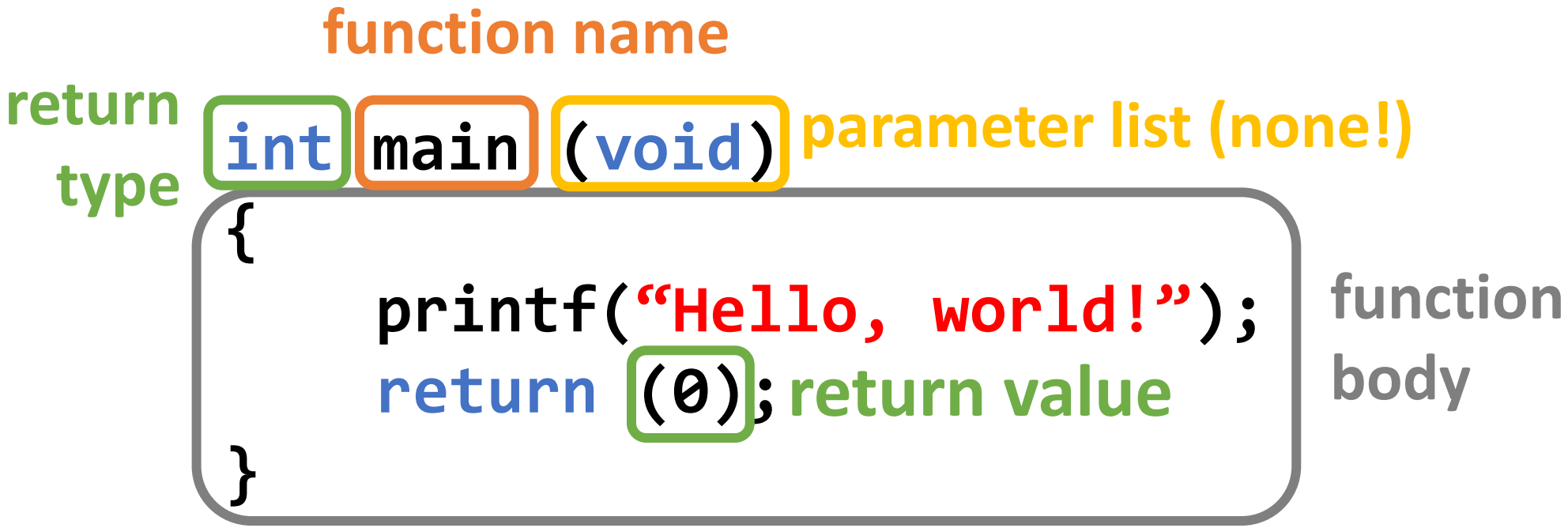
YES.

- A function need not accept input arguments
- A function need not return anything

© Alex Ufkes, 2020, 2021

Function Syntax

Remember the `main()` function?



Return value must match return type!

© Alex Ufkes, 2020, 2021

Function Syntax

In the general sense:

int, char, double, etc.

Function parameters

```
function_type function_name (input: type(s) name(s))
{
    /* Functi
       variab
       C statements  */

    return (result);     /* if needed */
}
```

Be consistent and descriptive

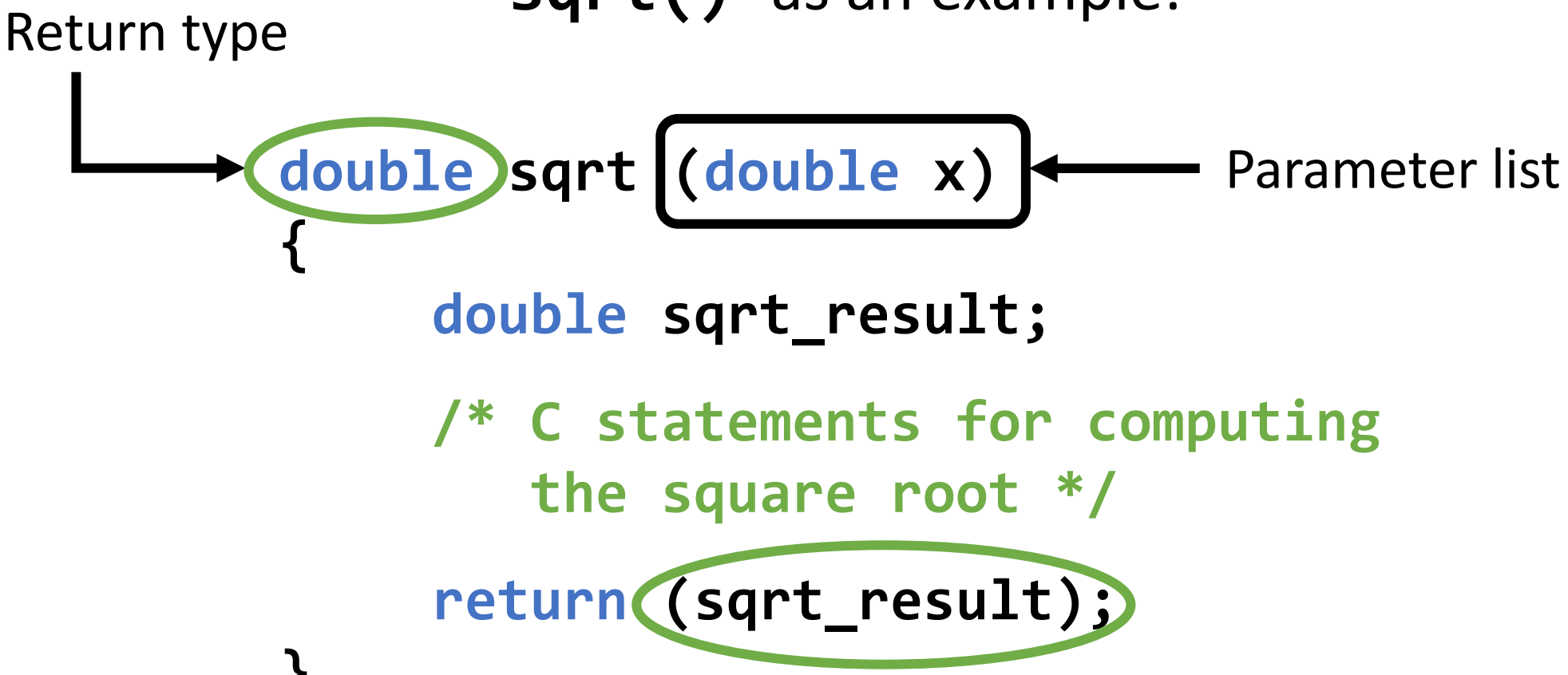
double x, int y, char c, etc.

result type must match function type!

© Alex Ufkes, 2020, 2021

Function Syntax

`sqrt()` as an example:

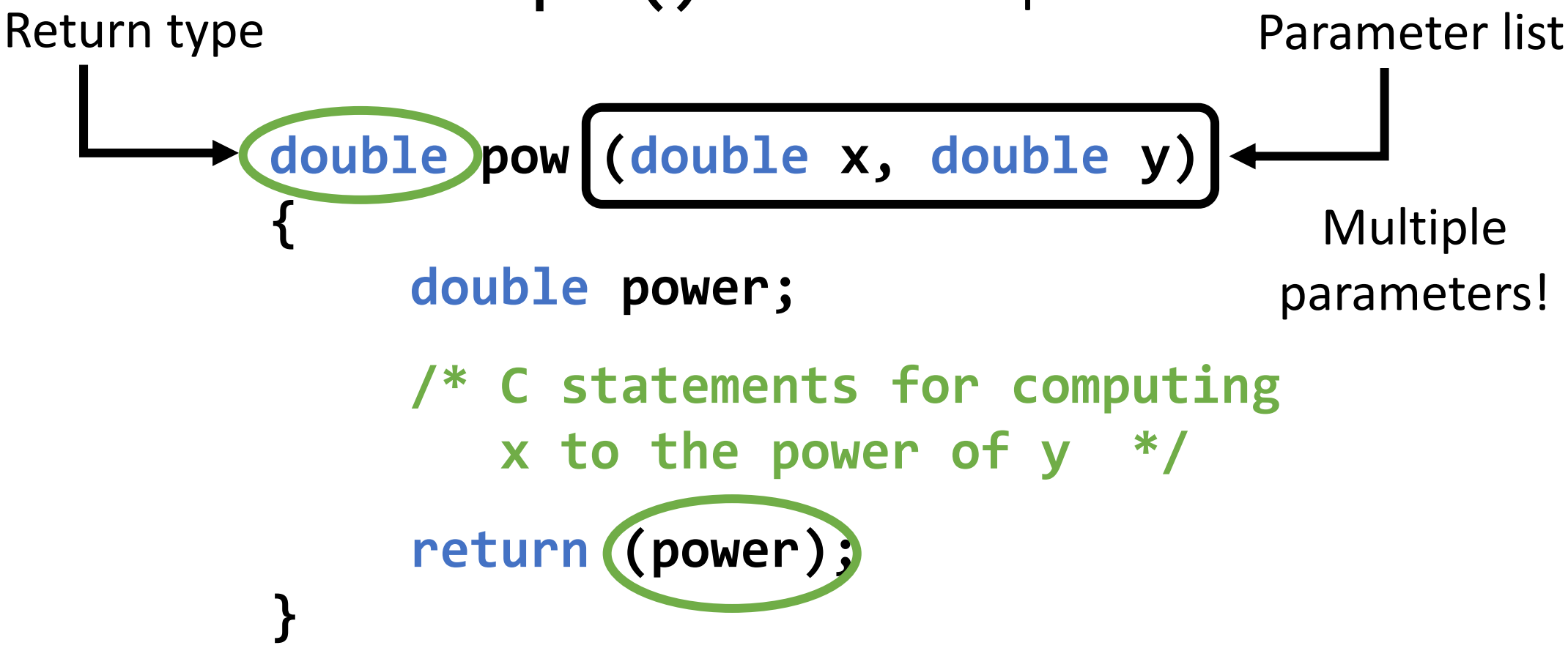



```
double sqrt (double x)
{
    double sqrt_result;

    /* C statements for computing
       the square root */

    return (sqrt_result);
}
```

© Alex Ufkes, 2020, 2021

Function Syntax

`pow()` as an example:



© Alex Ufkes, 2020, 2021

Calling a Function

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

int main (void)
{
      double y, x = 37.9;
      y = sqrt(x);
      printf("%lf", y);
}
```

How many function calls?

What are the arguments?

What are the return values?

© Alex Ufkes, 2020, 2021

User-Defined Functions

`sqrt()`, **`pow()`**, **`printf()`**, **`scanf()`** are all defined for us.

We gain access to them by including their respective header files.

What if we have our own task for which there is no existing function?

© Alex Ufkes, 2020, 2021

```c
#include <stdio.h>

int increment (int i)
{
    return i + 1;
}

int main (void)
{
    int x, k = 4;
    x = increment(k);
    printf("k+1 = %d\n", x);
    return 0;
}
```

User-defined function

Return value gets copied into **x**

Argument gets ***copied*** into parameter **i**.
k and **i** are **different** variables!

Calling user-defined function

© Alex Ufkes, 2020, 2021

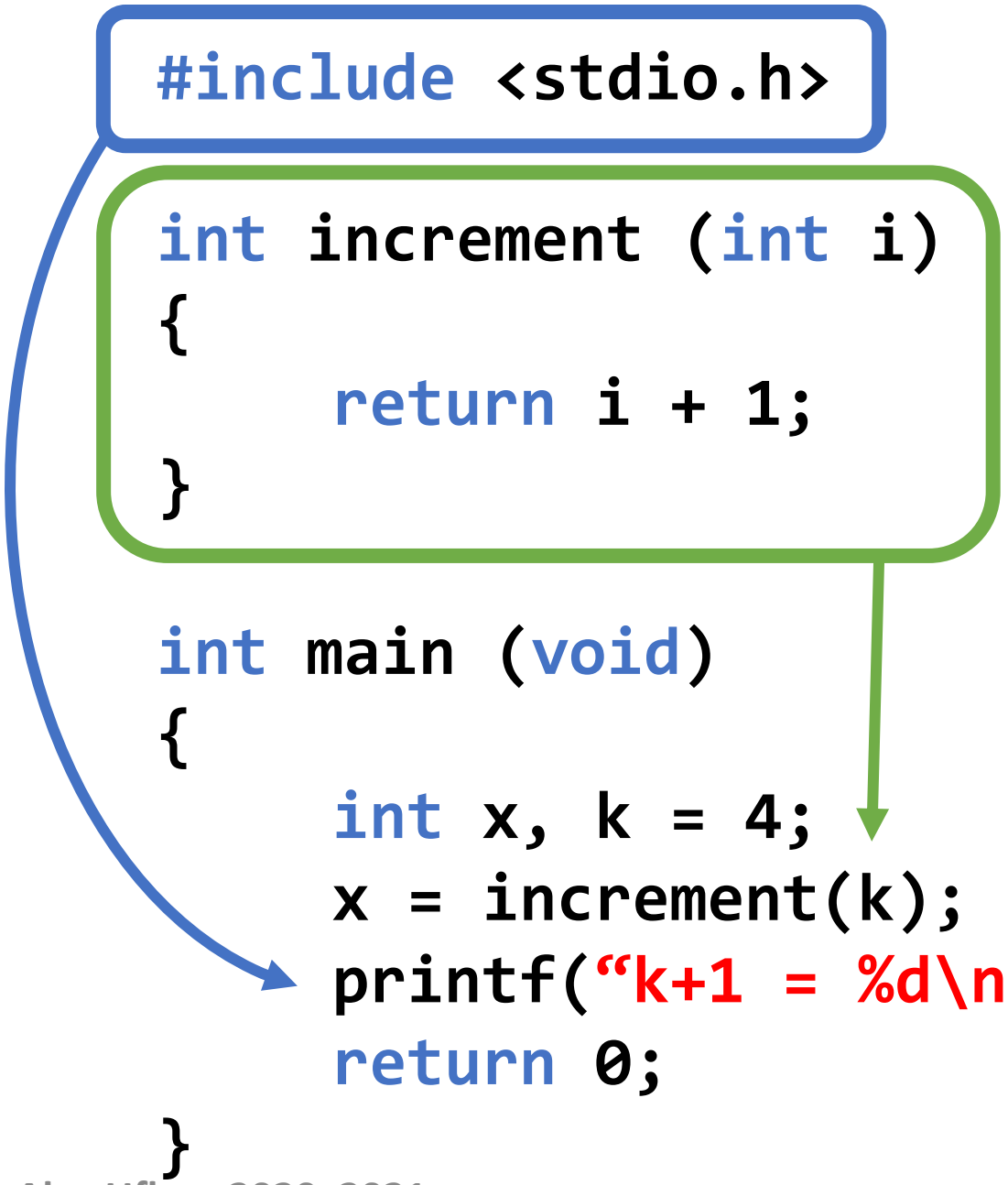
```c
#include <stdio.h>

int add_three (int a, int b, int c)
{
     int d = a + b + c;
     return d;
}

int main (void)
{
     printf("sum: %d\n", add_three(1, 2, 3));
     return 0;
}
```

User-defined function

Calling user-defined function

Arguments can be literals!

© Alex Ufkes, 2020, 2021



Functions must be declared before the code that makes the function call.

Otherwise we will get a compile error

© Alex Ufkes, 2020, 2021

```c
#include <stdio.h>

int increment (int i);

int main (void)
{
     int x, k = 4;
     x = increment(k);
     printf("k+1 = %d\n", x);
     return 0;
}

int increment (int i)
{
     return i + 1;
}
```

Solution!

Function *prototype:*
Tells the compiler that this function exists below the call, so don't panic.

Prototype must match function definition.

Problem?

© Alex Ufkes, 2020, 2021

`void` Functions

```
#include <stdio.h>

void stars (void)  /* No return value, no parameters */
{
     printf("**********\n");
}

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

Nothing to return, **`stars()`** does not need a **`return`** statement.

Since there are no parameters, we pass no arguments.

© Alex Ufkes, 2020, 2021

Something Useful?

- Function to say Hello?
- Function to compute hypotenuse?
- Function to find area of a circle?
- Function to find roots of a quadratic?

© Alex Ufkes, 2023

hello()

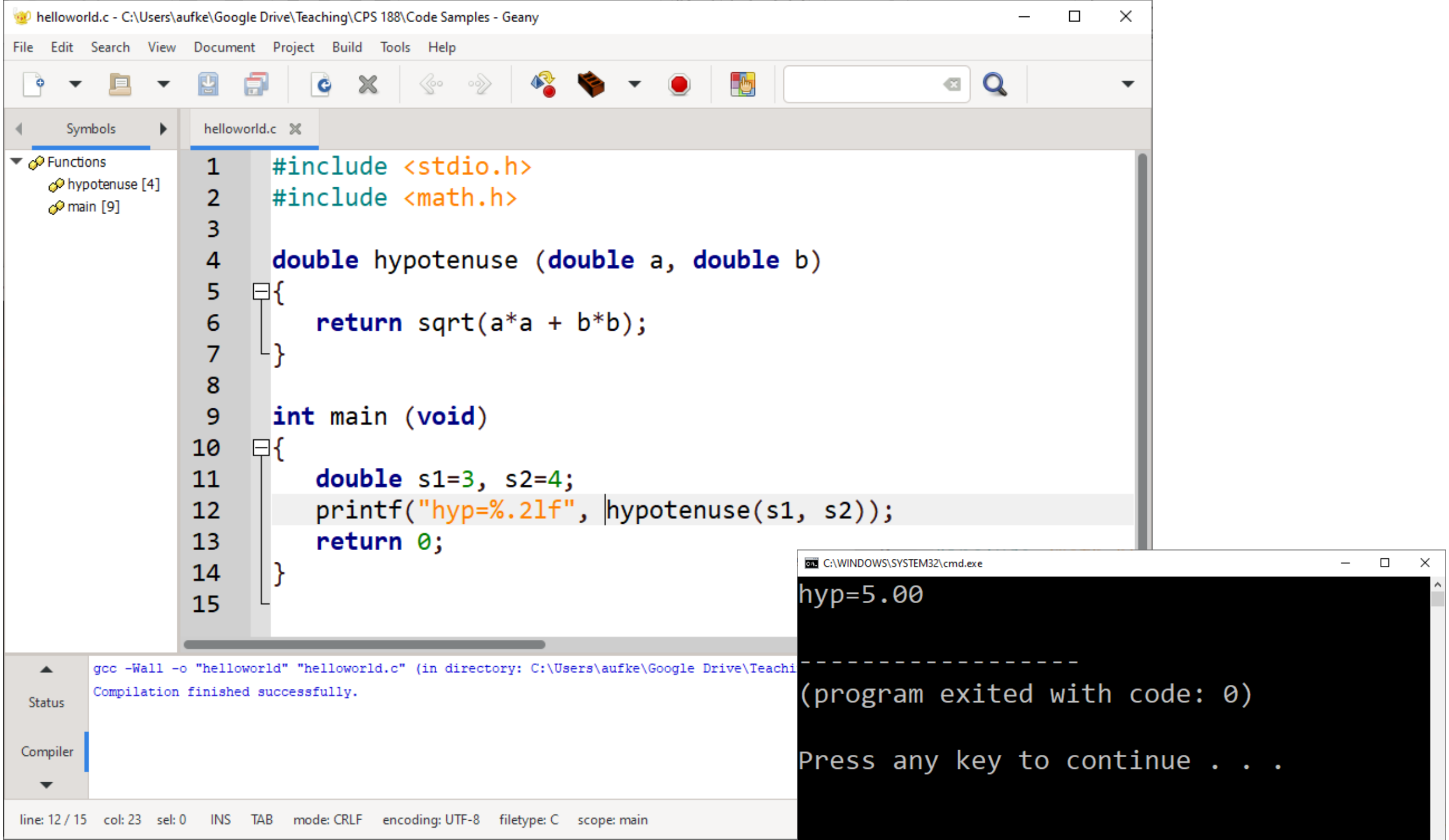
```
#include <stdio.h>

void hello (void)
{
    printf("Hello, world!\n");
}

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

© Alex Ufkes, 2023

hypotenuse()

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

double hypotenuse (double a, double b)
{
   return sqrt(a*a + b*b);
}

int main (void)
{
   double s1=3, s2=4;
   printf("hyp=%.2lf", hypotenuse(s1, s2));
   return 0;
}
```

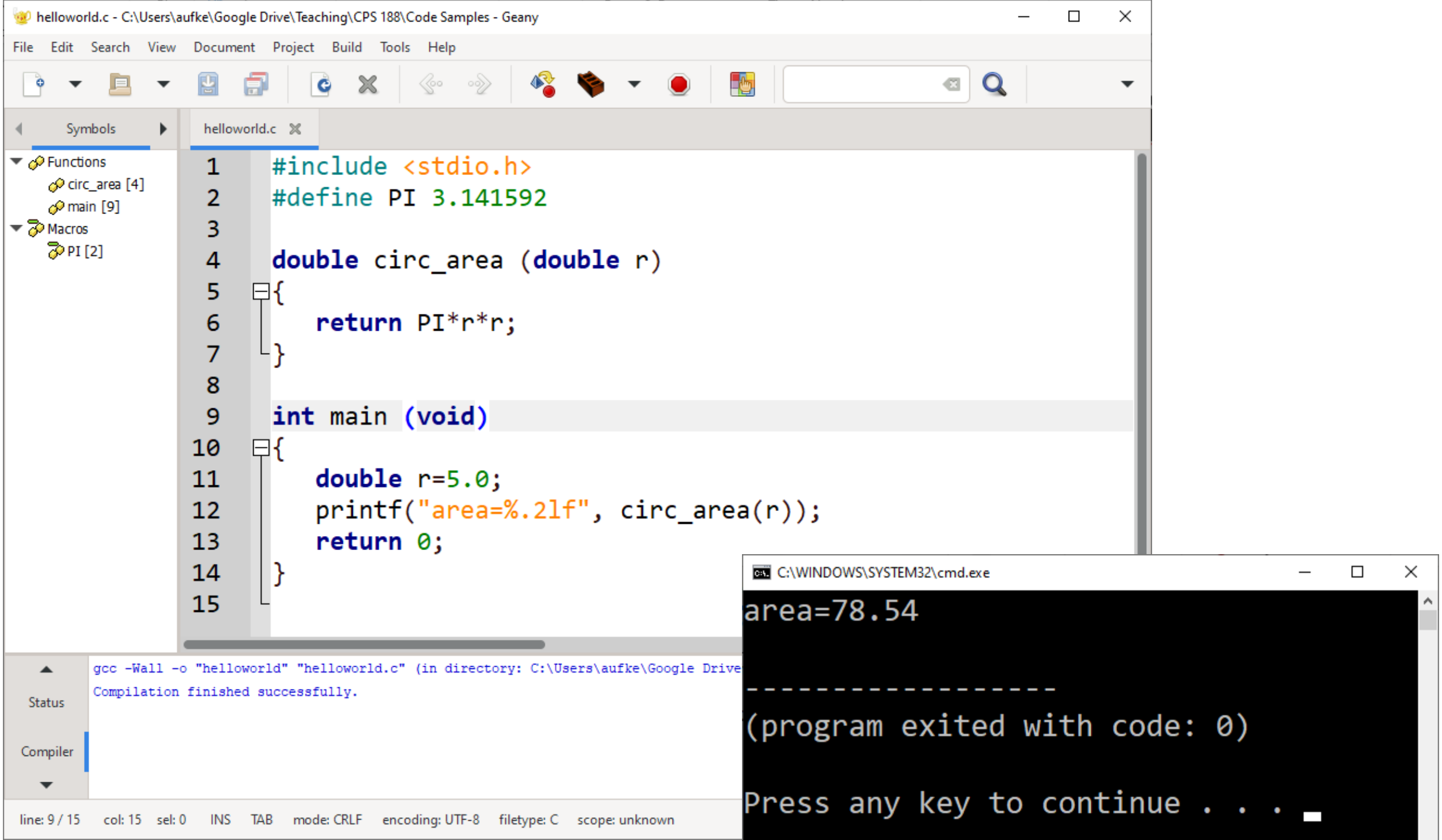
© Alex Ufkes, 2023


helloworld.c - C:\Users\aufke\Google Drive\Teaching\CPS 188\Code Samples - Geany
File Edit Search View Document Project Build Tools Help
Symbols
helloworld.c
Functions
hypotenuse [4]
main [9]
1 #include <stdio.h>
2 #include <math.h>
3
4 double hypotenuse (double a, double b)
5 {
6 return sqrt(a*a + b*b);
7 }
8
9 int main (void)
10 {
11 double s1=3, s2=4;
12 printf("hyp=%.2lf", hypotenuse(s1, s2));
13 return 0;
14 }
15
gcc -Wall -o "helloworld" "helloworld.c" (in directory: C:\Users\aufke\Google Drive\Teachi
Compilation finished successfully.
Status
Compiler
line: 12 / 15 col: 23 sel: 0 INS TAB mode: CRLF encoding: UTF-8 filetype: C scope: main
C:\WINDOWS\SYSTEM32\cmd.exe
hyp=5.00
(program exited with code: 0)
Press any key to continue . . .

circ_area()

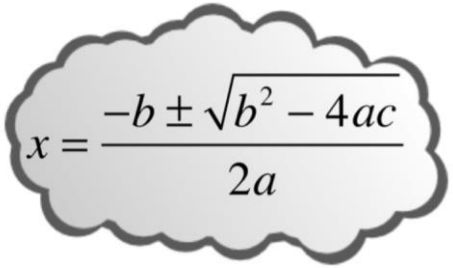
```
#include <stdio.h>
#define PI 3.141592

double circ_area (double r)
{
    return PI*r*r;
}

int main (void)
{
    double r=5.0;
    printf("area=%.2lf", circ_area(r));
    return 0;
}
```

© Alex Ufkes, 2023


helloworld.c - C:\Users\aufke\Google Drive\Teaching\CPS 188\Code Samples - Geany
File Edit Search View Document Project Build Tools Help
Symbols
helloworld.c
Functions
circ_area [4]
main [9]
Macros
PI [2]
1 #include <stdio.h>
2 #define PI 3.141592
3
4 double circ_area (double r)
5 {
6 return PI*r*r;
7 }
8
9 int main (void)
10 {
11 double r=5.0;
12 printf("area=%.2lf", circ_area(r));
13 return 0;
14 }
15
gcc -Wall -o "helloworld" "helloworld.c" (in directory: C:\Users\aufke\Google Drive
Compilation finished successfully.
Status
Compiler
line: 9 / 15 col: 15 sel: 0 INS TAB mode: CRLF encoding: UTF-8 filetype: C scope: unknown
C:\WINDOWS\SYSTEM32\cmd.exe
area=78.54
(program exited with code: 0)
Press any key to continue . . .

quad()

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

double quad (double a, double b, double c)
{
    double disc = sqrt(b*b – 4*a*c);
    double root1 = (-b + tmp)/(2*a);
    double root2 = (-b - tmp)/(2*a);
    return root1; // What about root2?
}

int main (void)
{
    double c1=1, c2=2, c3=3;
    double r1 = quad(c1, c2, c3);
    printf(“root1 = %.2lf”, r1);
    return 0;
}
```



“Can we have multiple results?”

NO.

We’re stuck!

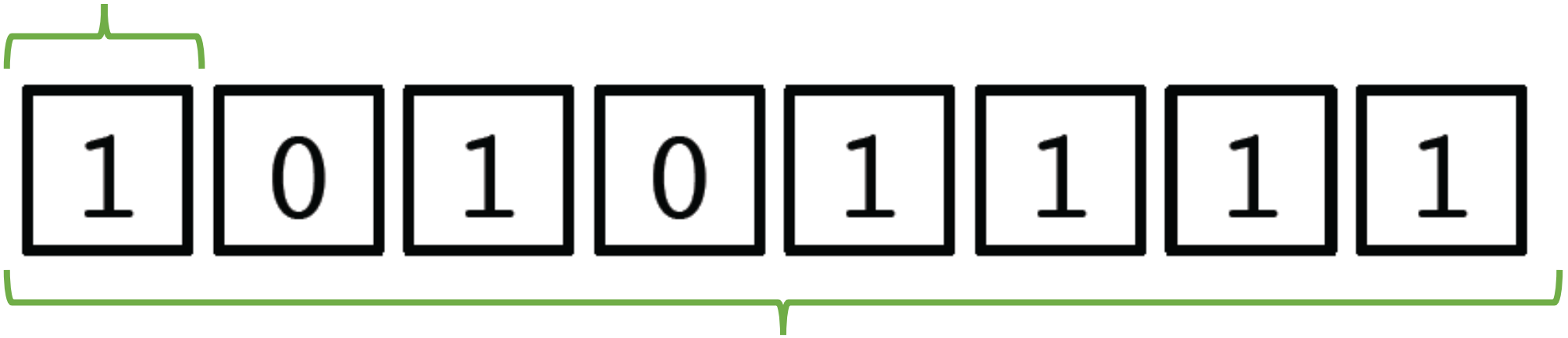
- One option? Two separate functions, one for each root.
- Another option? Pointers, which we haven’t learned yet.

© Alex Ufkes, 2023

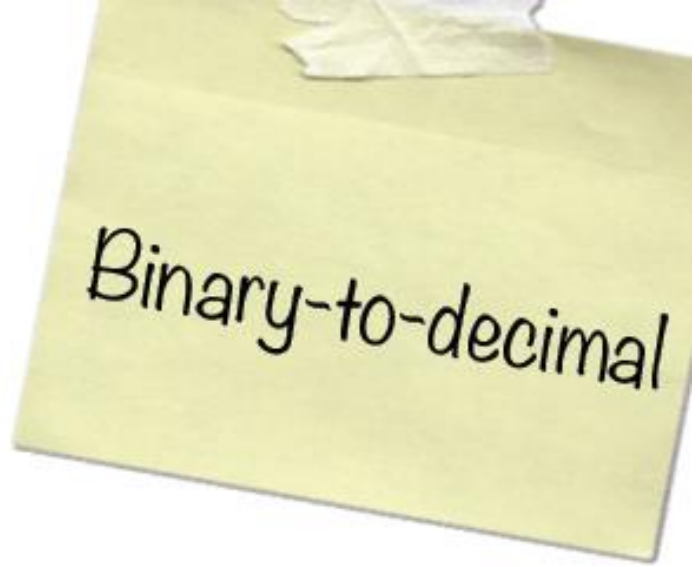
Bitwise Operations

Recall: Binary Numbers

Bit (0 or 1)



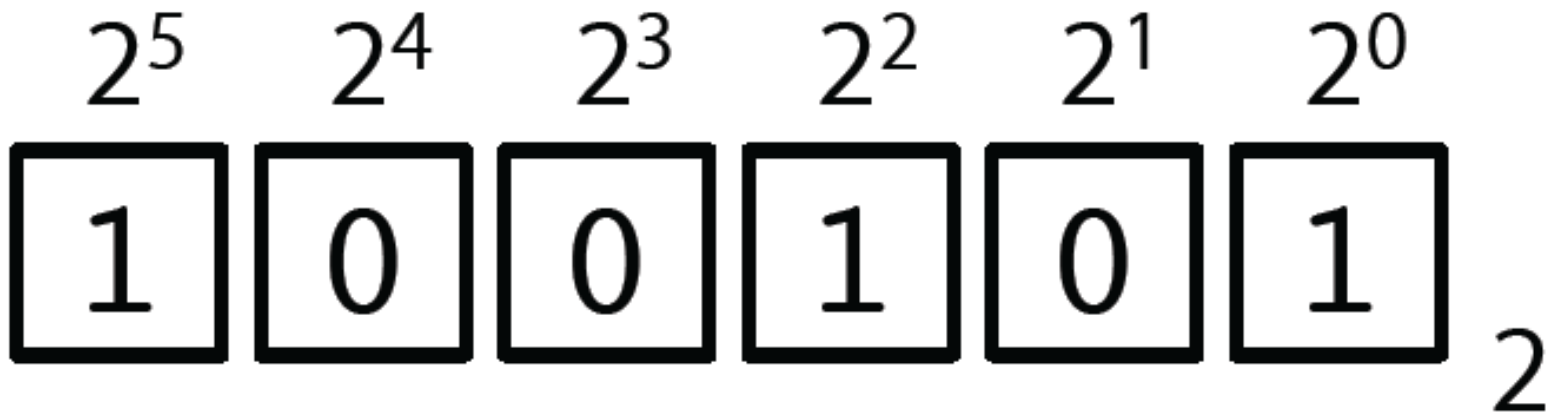
Byte (group of 8 bits)



Binary: Base **2** number system

2^5	2^4	2^3	2^2	2^1	2^0	
1	0	0	1	0	1	2

Indicates base



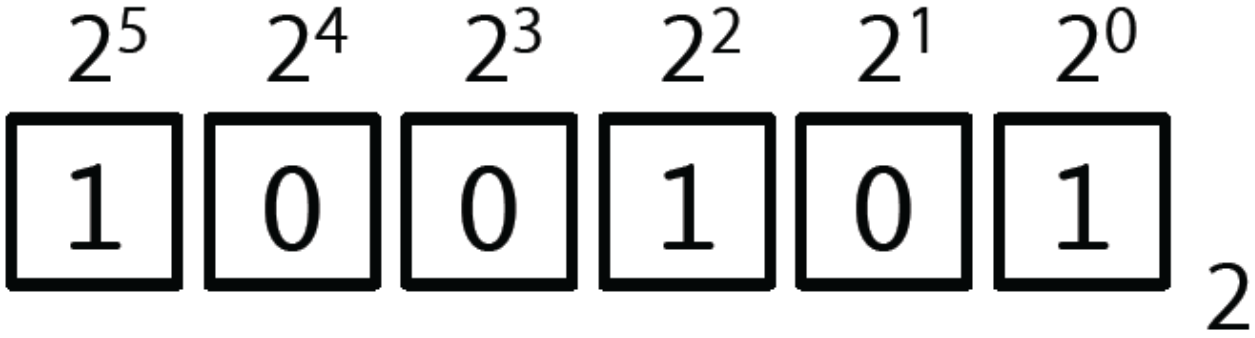
$1 \times 2^0 = 1$
$0 \times 2^1 = 0$
$1 \times 2^2 = 4$
$0 \times 2^3 = 0$
$0 \times 2^4 = 0$
$1 \times 2^5 = 32$

37

Integers

- In practice, integers can be 8, 16, 32, or 64 bits.
- We most commonly deal with 32 bit integers
- This is the size of a standard **`int`** in C

Recall:



Only 6 bits here
How to convert to 32?

Add leading zeroes: 00000000000000000000000000100101

In base 10, 578 is the same number as 000000578. Same applies in binary.

Bitwise Operators

- Arithmetic operators operate on individual numeric values
- These values are represented as a sequence of binary bits
- Bitwise operators operate on individual, corresponding bits

```
&     Bitwise AND
|     Bitwise OR
^     Bitwise XOR (exclusive OR)
~     Bitwise complement
>>    Bit shift right
<<    Bit shift left
```

Bitwise Operators: Truth Tables

&

X	Y	X & Y
0	0	0
0	1	0
1	0	0
1	1	1

|

X	Y	X | Y
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise Arithmetic

- No carry the one, no remainder, no nothing.
- Just a simple operation between corresponding bits.

```
  00101101
& 10110111
----------
  00100101
```

```
  00101101
| 10110111
----------
  10111111
```

Bitwise Operators: Truth Tables

X	Y	X ^ Y
0	0	0
0	1	1
1	0	1
1	1	0

X	~X
0	1
1	0

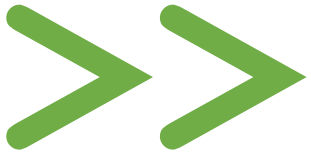
Bitwise Arithmetic

- No carry the one, no remainder, no nothing.
- Just a simple operation between corresponding bits.

```
  00101101
^ 10110111
----------
  10011010
```

```
~ 10110111
----------
  01001000
```

Bitwise Operators: Bit Shifting



X	Y	X >> Y
01010	1	00101
01010	2	00010
01010	3	00001
01010	4	00000

Maintain highest order bit. Negative numbers stay negative!

X	Y	X << Y
00001010	1	00010100
00001010	2	00101000
00001010	3	01010000
00001010	4	10100000

Introduce trailing zeroes

Bit Masking

- Bit masking can be used to access individual bits.
- This is incredibly common at the hardware register level.
- Registers (on the CPU) are *very* fast, and *very* few in number.
- In embedded systems, even main memory can be extremely limited.
- It becomes important to use memory as efficiently as we can.
- We don't think twice about using an **`int`** to represent the number 57.
- However, 57 can be represented in ¼ the size using a **`char`**.
- 32 bits VS 8 bits

Example: Get n^{th} bit

Here, the rightmost bit is considered 0^{th} bit

Quincy 2005 - [main.c]

File Edit View Project Debug Tools Window Help

```c
#include<stdio.h>

int nthBit(int num, int n)
{
    return (num >> n) & 1;
}

int main(void)
{
    int x = 27;   // 00011011

    printf("bit 0 = %d\n", nthBit(x, 0));
    printf("bit 1 = %d\n", nthBit(x, 1));
    printf("bit 2 = %d\n", nthBit(x, 2));
    printf("bit 3 = %d\n", nthBit(x, 3));
    printf("bit 4 = %d\n", nthBit(x, 4));

    return 0;
}
```

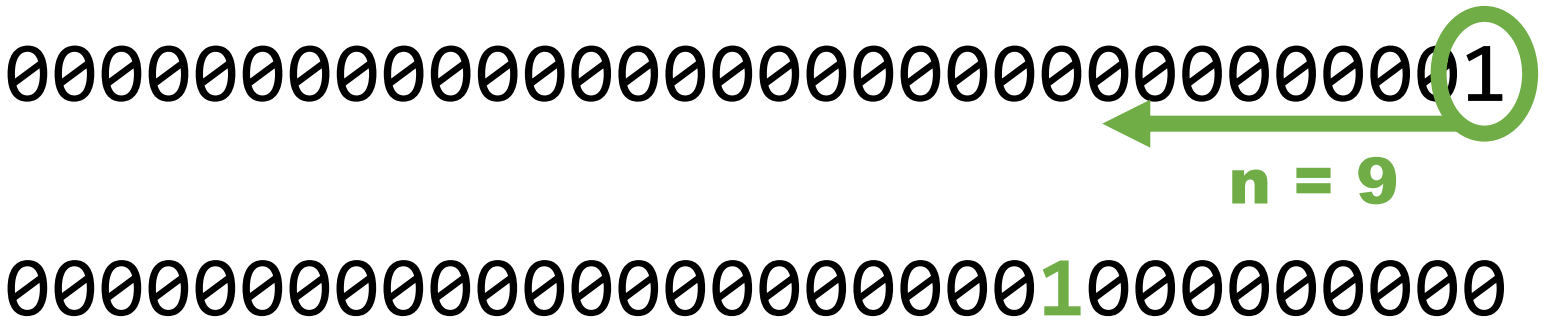
Quincy 2005

```
bit 0 = 1
bit 1 = 1
bit 2 = 0
bit 3 = 1
bit 4 = 1

Press Enter to return to Quincy...
```

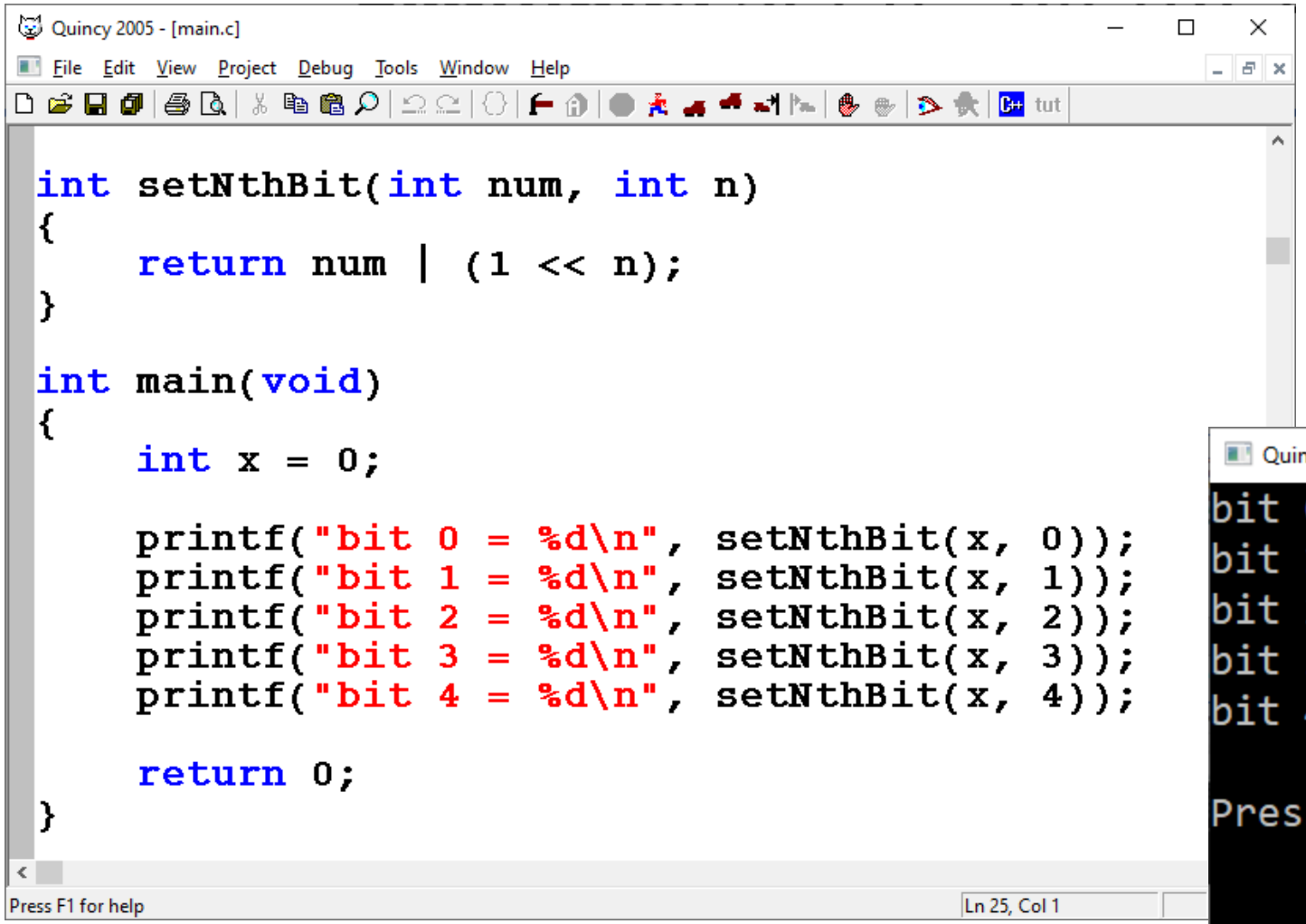
Example: Set n^{th} bit (to 1)

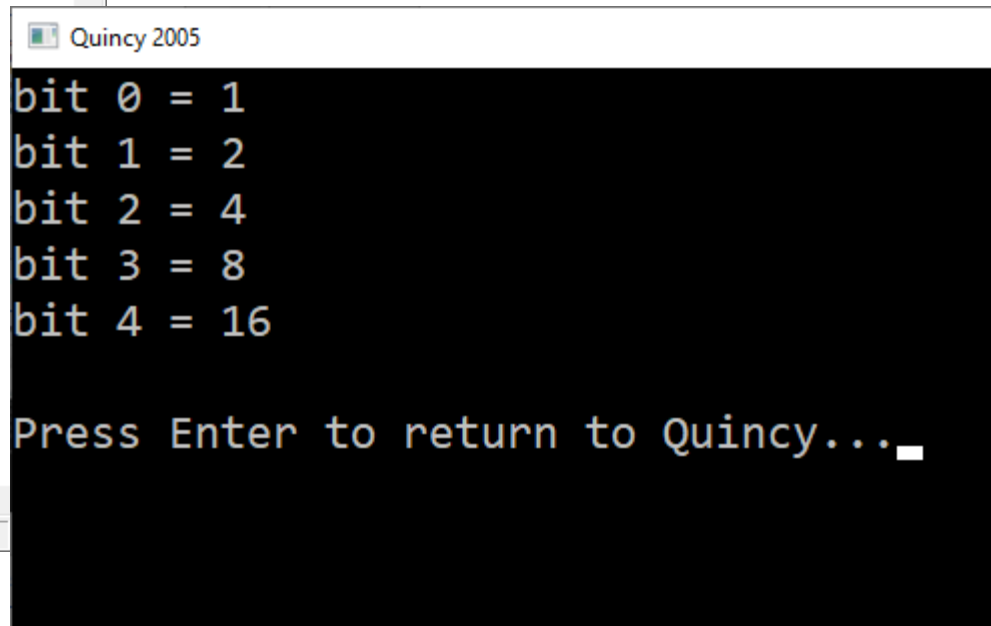
```
x = x | (1 << n);
```

- Start with 1, shift to the left n positions.
- OR with x. n^{th} bit becomes 1, all others stay the same



Example: Set n^{th} bit (to 1)





Example: Toggle n^{th} bit (invert)

If 1, change to 0. If 0, change to 1:

- Simple change, use XOR instead of OR.

Quincy 2005 - [main.c]

File Edit View Project Debug Tools Window Help

```
int toggleNthBit(int num, int n)
{
    return num ^ (1 << n);
}

int main(void)
{
    int x = 27; //  00011011

    printf("bit 0 = %d\n", toggleNthBit(x, 0));
    printf("bit 1 = %d\n", toggleNthBit(x, 1));
    printf("bit 2 = %d\n", toggleNthBit(x, 2));
    printf("bit 3 = %d\n", toggleNthBit(x, 3));
    printf("bit 4 = %d\n", toggleNthBit(x, 4));

    return 0;
}
```

Quincy 2005

```
bit 0 = 26
bit 1 = 25
bit 2 = 31
bit 3 = 19
bit 4 = 11

Press Enter to return to Quincy...
```

Example: Clear n^{th} bit (set to 0)

```
x = x & (~(1 << n));
```

- Start with 1, shift to the left n positions.
- Invert and AND with x. n^{th} bit becomes zero

```
00000000000000000000000000000001
00000000000000000000000010000000
11111111111111111111111101111111
```

Example: Clear n^{th} bit

- Start with 1, shift to the left n positions.
- Invert and AND with x. n^{th} bit becomes zero

Quincy 2005 - [main.c]

File Edit View Project Debug Tools Window Help

```
int clearNthBit(int num, int n)
{
    return num & ~(1 << n);
}

int main(void)
{
    int x = 27; //  00011011

    printf("bit 0 = %d\n", clearNthBit(x, 0));
    printf("bit 1 = %d\n", clearNthBit(x, 1));
    printf("bit 2 = %d\n", clearNthBit(x, 2));
    printf("bit 3 = %d\n", clearNthBit(x, 3));
    printf("bit 4 = %d\n", clearNthBit(x, 4));

    return 0;
}
```

Quincy 2005

```
bit 0 = 26
bit 1 = 25
bit 2 = 27
bit 3 = 19
bit 4 = 11

Press Enter to return to Quincy...
```

Questions?

© Alex Ufkes, 2023