CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 6: Pointers

Toronto Metropolitan University

Notice!

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

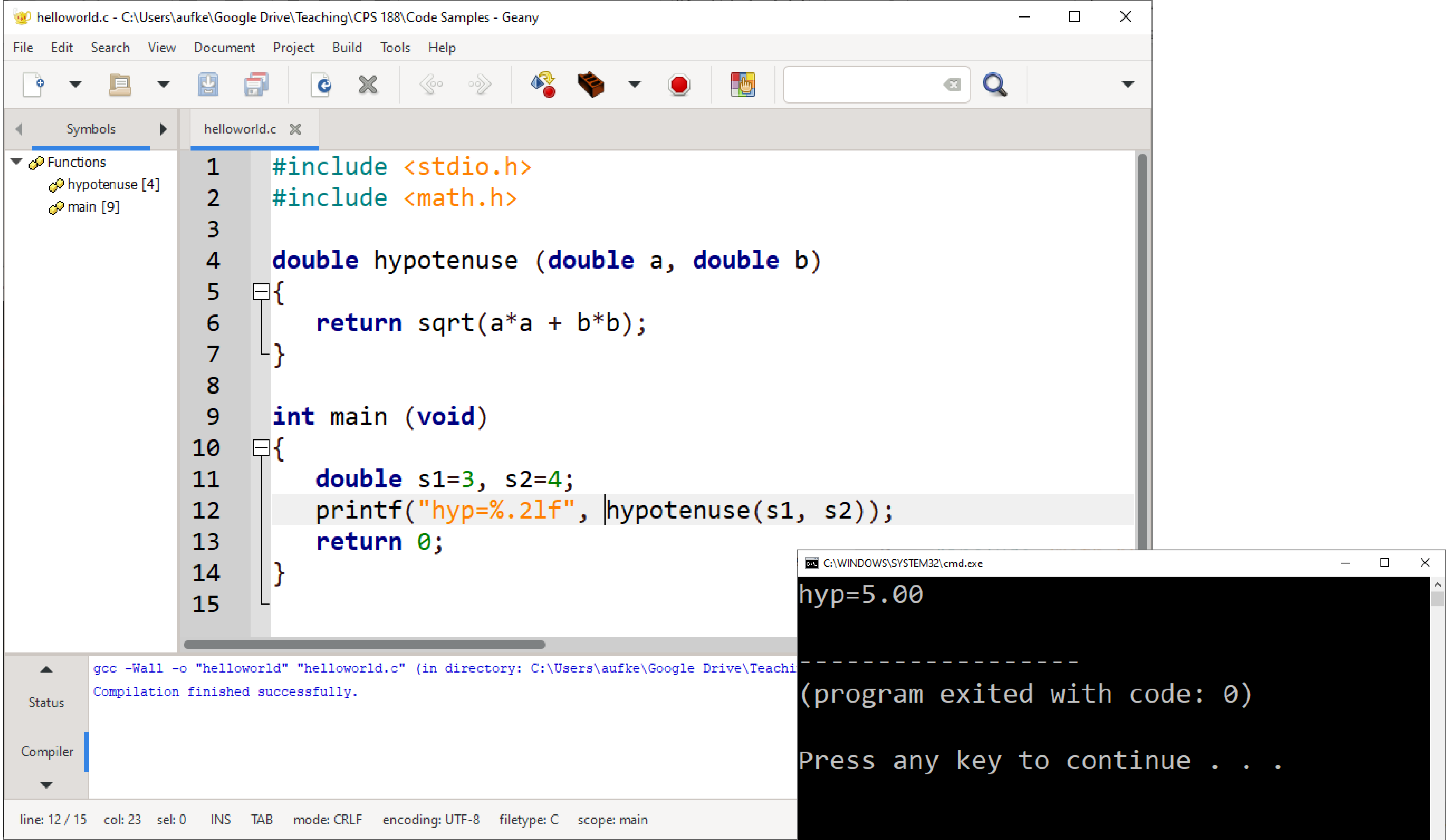
Previously:

Useful Functions

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

hypotenuse()

```
#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;
}
```



```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;
}
```

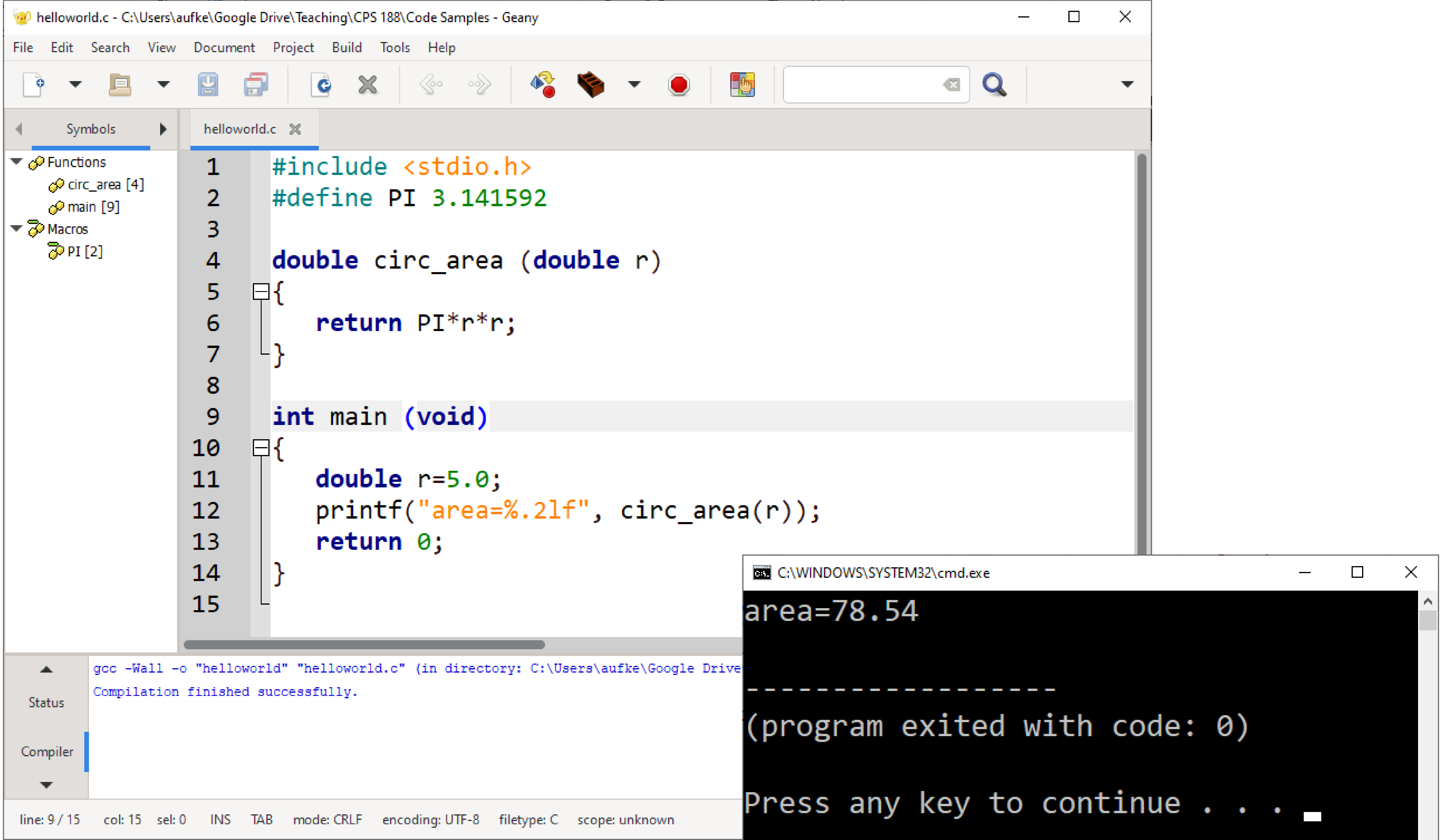
```
gcc -Wall -o "helloworld" "helloworld.c" (in directory: C:\Users\aufke\Google Drive\Teachi
Compilation finished successfully.
```

```
hyp=5.00

------------------
(program exited with code: 0)

Press any key to continue . . .
```

line: 12 / 15   col: 23   sel: 0   INS   TAB   mode: CRLF   encoding: UTF-8   filetype: C   scope: main

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;
}
```



```c
#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;
}

```

```
gcc -Wall -o "helloworld" "helloworld.c" (in directory: C:\Users\aufke\Google Drive
Compilation finished successfully.
```

line: 9 / 15   col: 15   sel: 0   INS   TAB   mode: CRLF   encoding: UTF-8   filetype: C   scope: unknown

```
area=78.54

------------------
(program exited with code: 0)

Press any key to continue . . .
```

quad()

```
#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 + disc)/(2*a);
   double root2 = (-b - disc)/(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;
}
```

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

“Can we have multiple results?”

NO.

We’re stuck!

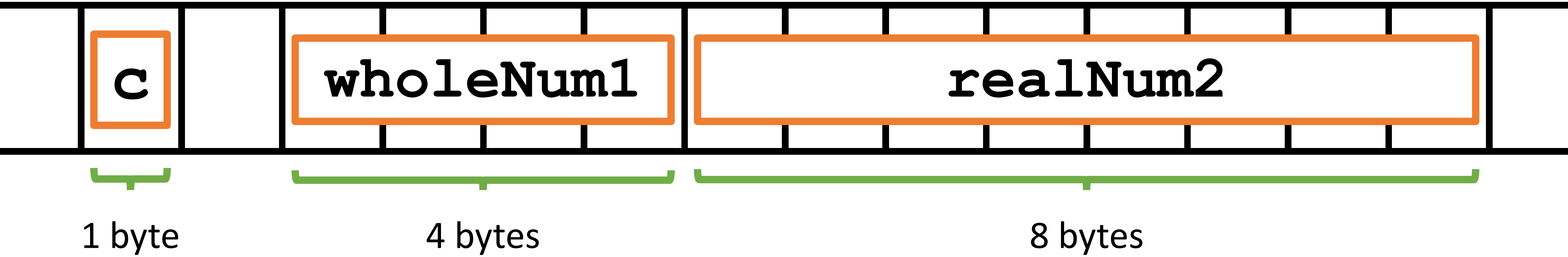
- One option? Two separate functions, one for each root.
- Another option? Pointers, which we haven’t learned yet.

Moving On…

POINTERS

Recall: Variables in Memory

```
char c;                 /* 1 byte allocated  */
int wholeNum1;          /* 4 bytes allocated */
double realNum2;        /* 8 bytes allocated */
```



Recall: Use & to get address

The & operator returns the address of a variable

```
int number
scanf(“%d”, &number);
```

Pointer Data Types

A **pointer** is a variable that stores the **address** of another variable:

```
char *cptr;     /* stores the address of a char   */
int *iptr;      /* stores the address of an int   */
float *fptr;    /* stores the address of a float  */
double *dptr    /* stores the address of a double */
```

The * is used in the variable declaration to specify a pointer.

```
char *cptr;      /* stores the address of a char   */
int *iptr;       /* stores the address of an int   */
float *fptr;     /* stores the address of a float  */
double *dptr     /* stores the address of a double */
```

In a 32-bit application, pointers are also 32 bits (4 bytes), regardless of the data type they are pointing to.

char* is 4 bytes, even though it points to a **char**, which is 1 byte.

The value of a pointer is simply a memory location (address)

Declaration Syntax

`x` is a pointer, `y` and `z` are integers

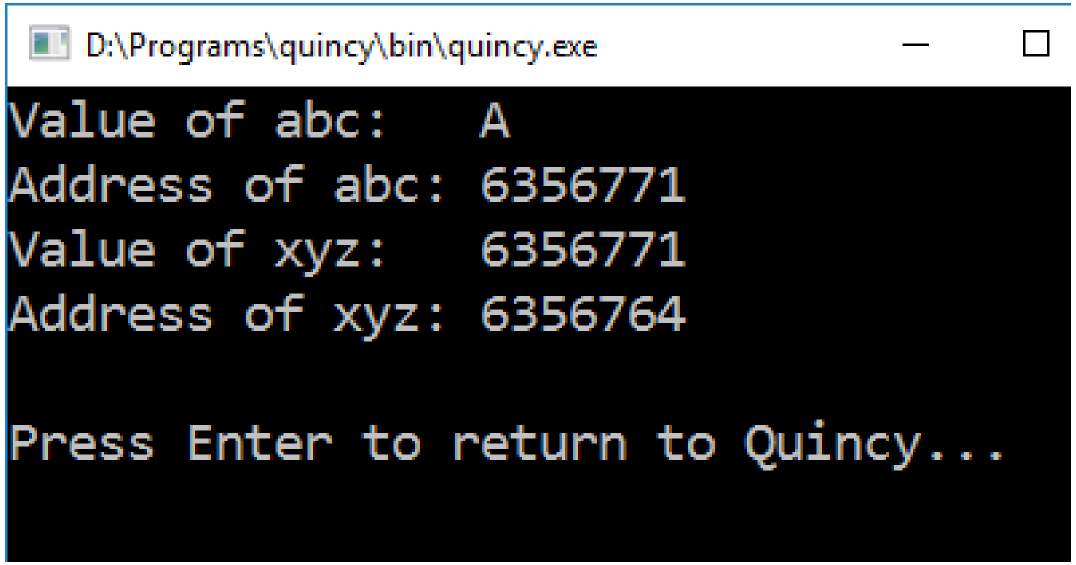
```
int* x, y, z;
```

Each variable needs a * if it’s a pointer

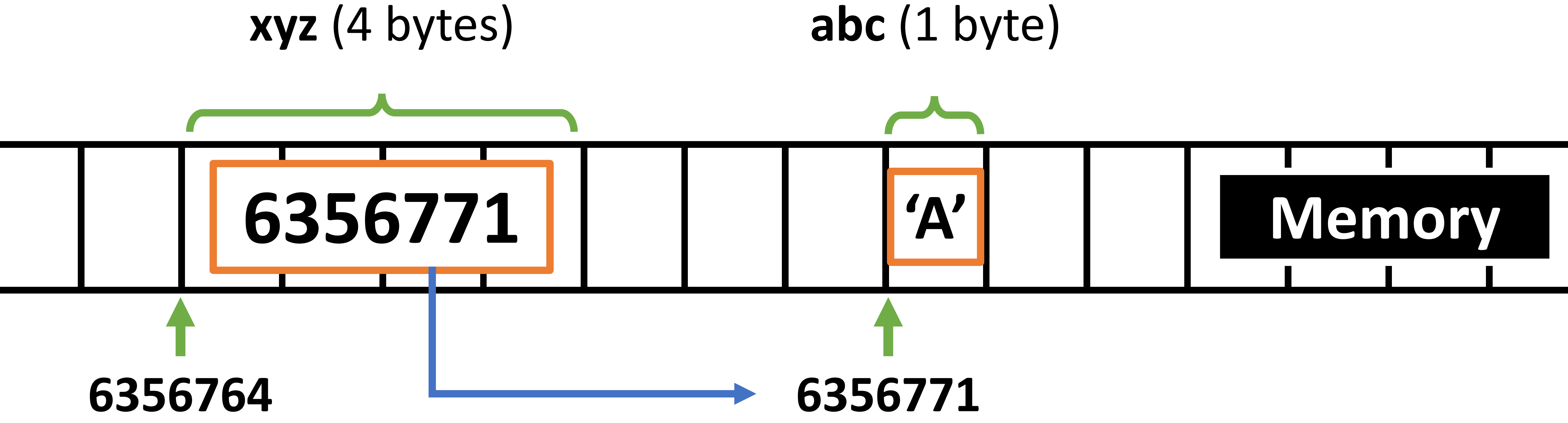
```
int* x, * y, * z;
```

```
/* Easiest, most clear */
int *x, *y, *z;
```

```
char abc = 'A';
char *xyz = &abc;

printf("Value of abc:   %c\n", abc);
printf("Address of abc: %d\n", &abc);
printf("Value of xyz:   %d\n", xyz);
printf("Address of xyz: %d\n", &xyz);
```



```
Value of abc:   A
Address of abc: 6356771
Value of xyz:   6356771
Address of xyz: 6356764

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



Dereferencing

*

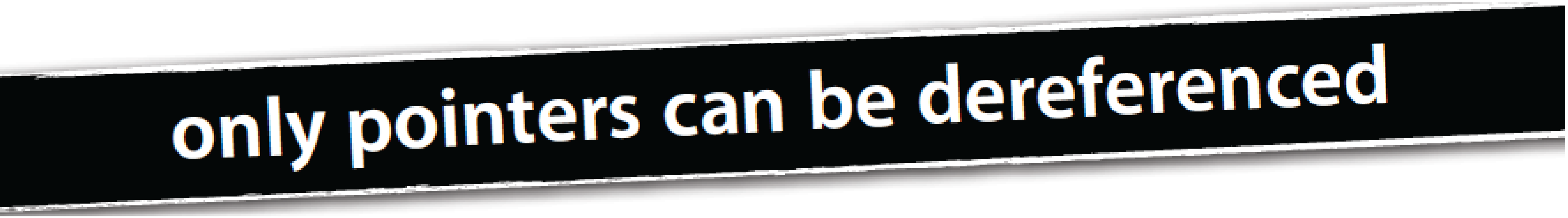
* Operator

NOT multiplication! - Different in the context of pointers.

Use * to ***dereference*** a pointer.

Dereferencing is used to access the memory location being “pointed” to.

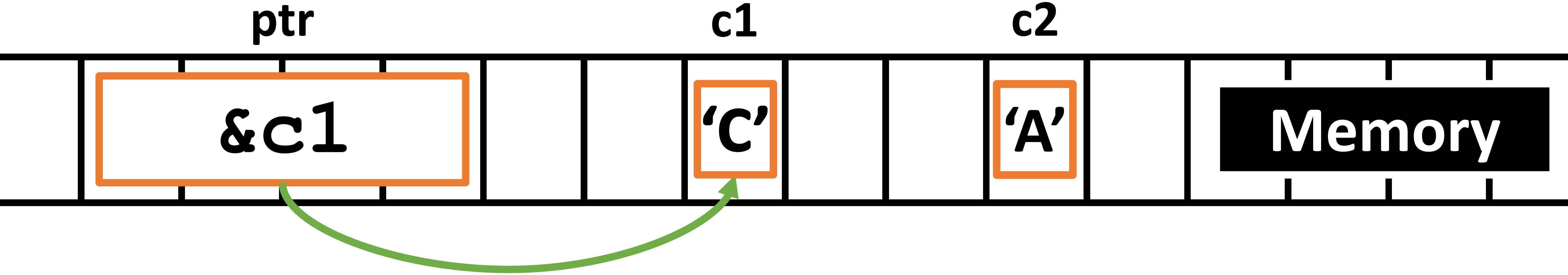
Primitive data types (int, char, float, double) **CANNOT** be dereferenced



only pointers can be dereferenced

```
char c1 = ‘A’, c2;
char *ptr = &c1;      // ptr stores the address of c1
c2 = *ptr;            // dereference ptr, store in c2
*ptr = ‘C’;           // store ‘C’ at location being
                      // pointed to by ptr
```

```
printf(“c1 is:   %c\n”, c1);       /* c1 is:   C  */
printf(“*ptr is: %c\n”, *xyz);     /* *ptr is: C  */
printf(“c2 is:   %c\n”, c2);       /* c2 is:   A  */
```



Dereferencing Confusion

```
char abc = 'A';
char *xyz = &abc;    /* NOT dereferencing! */
*xyz = ‘$’;          /* THIS is dereferencing! */
```

Context matters!

1) In the context of declaration, the * indicates we are declaring a pointer.

2) Outside of declarations, the * indicates we are dereferencing a pointer.

3) In the context of a binary operation (two operands), the * indicates multiplication.

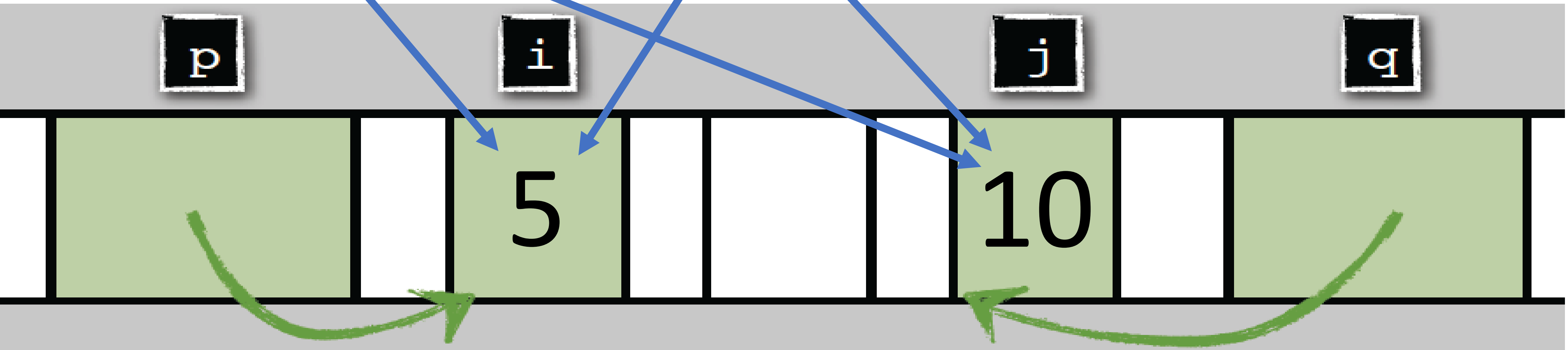
It Gets Complicated…

```
int i, j, *p, *q;    // pointers and ints together
p = &i;              // value of p is the address of i
q = &j;              // value of q is the address of j
*p = 5;              // store 5 at the address stored in p
*q = *p + i;         // dereference p, add i, store at
                     // address stored in q

printf(“i = %d, j = %d\n”, i, j);
printf(“i = %d, j = %d\n”, *p, *q);
```

Output?

```
i = 5, j = 10
```

```
int i, j, *p, *q;     // pointers and ints together
p = &i;               // value of p is the address of i
q = &j;               // value of q is the address of j
*p = 5;               // store 5 at the address in p
*q = *p + i;          // dereference p, add i, store at
                      // address in q

printf(“i = %d, j = %d\n”, i, j);
```



You Need Help!

More Pointer Examples

```c
int x = 57, y = 0;
int *a = &y, *b = &x;

*a = 12;      // y = 12
b = a;        // b and a both point to y
*b = 15;      // y = 15

printf(“x = %d, y = %d\n", x, y);
```

```
x = 57, y = 15
```

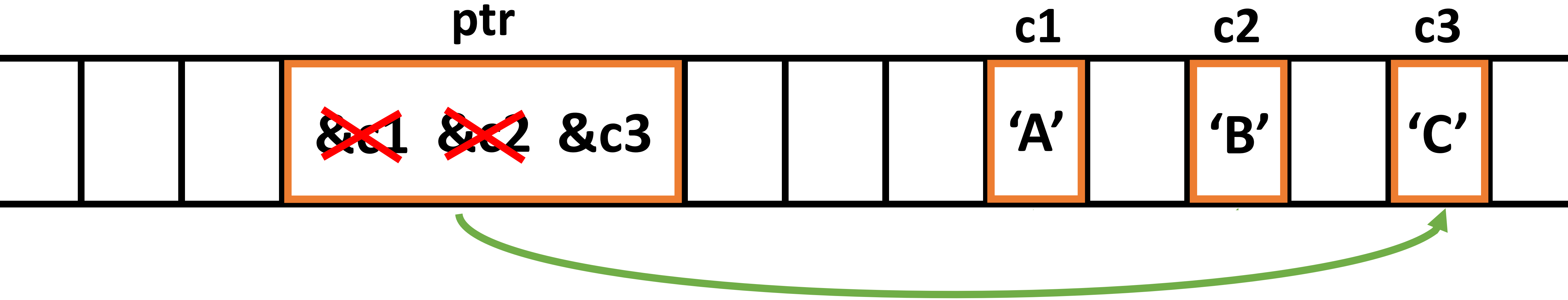
```
int x = 57, y = 0;
int *a, *b = &y;

*b = 7;          // y = 7
a = &x;          // a points to x
x = *a - *b;     // x = x – y

printf(“x = %d, y = %d\n", x, y);
```

```
x = 50, y = 7
```

```
int x, y, z, *p1, *p2, *p3, *p4;

p1 = &x;      // p1 points to x
p4 = p1;      // p4 points to x
p2 = p4;      // p2 points to x
*p4 = 5;      // x = 5
y = x;        // y = 5

printf(“x = %d, y = %d\n", x, y);
```

```
x = 5, y = 5
```

```
char c1, c2, c3, *ptr;

ptr = &c1;         // value of ptr is the address of c1
*ptr = ‘A’;        // dereference ptr, store ‘A’

ptr = &c2;         // value of ptr is the address of c2
*ptr = ‘B’;        // dereference ptr, store ‘B’

ptr = &c3;         // value of ptr is the address of c3
*ptr = ‘C’;        // dereference ptr, store ‘C’
```



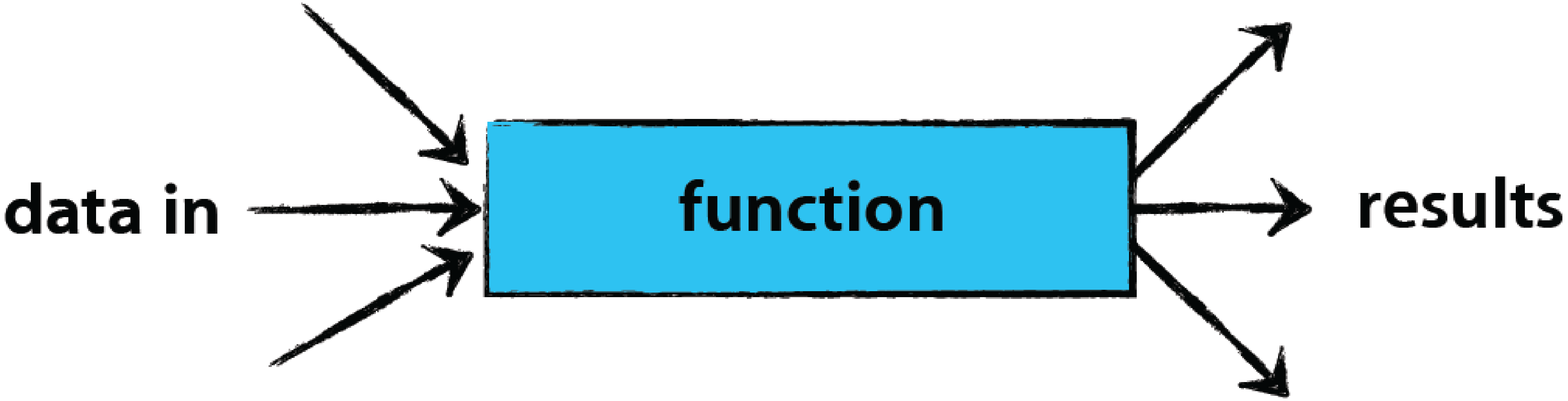
Is This OK?

```
int x, *p1;
p1 = &x;

printf("Please enter an integer: ");
scanf("%d", p1); /* Missing & */
printf("x = %d\n", x);
```

Yes! scanf needs an address, p1 is an address

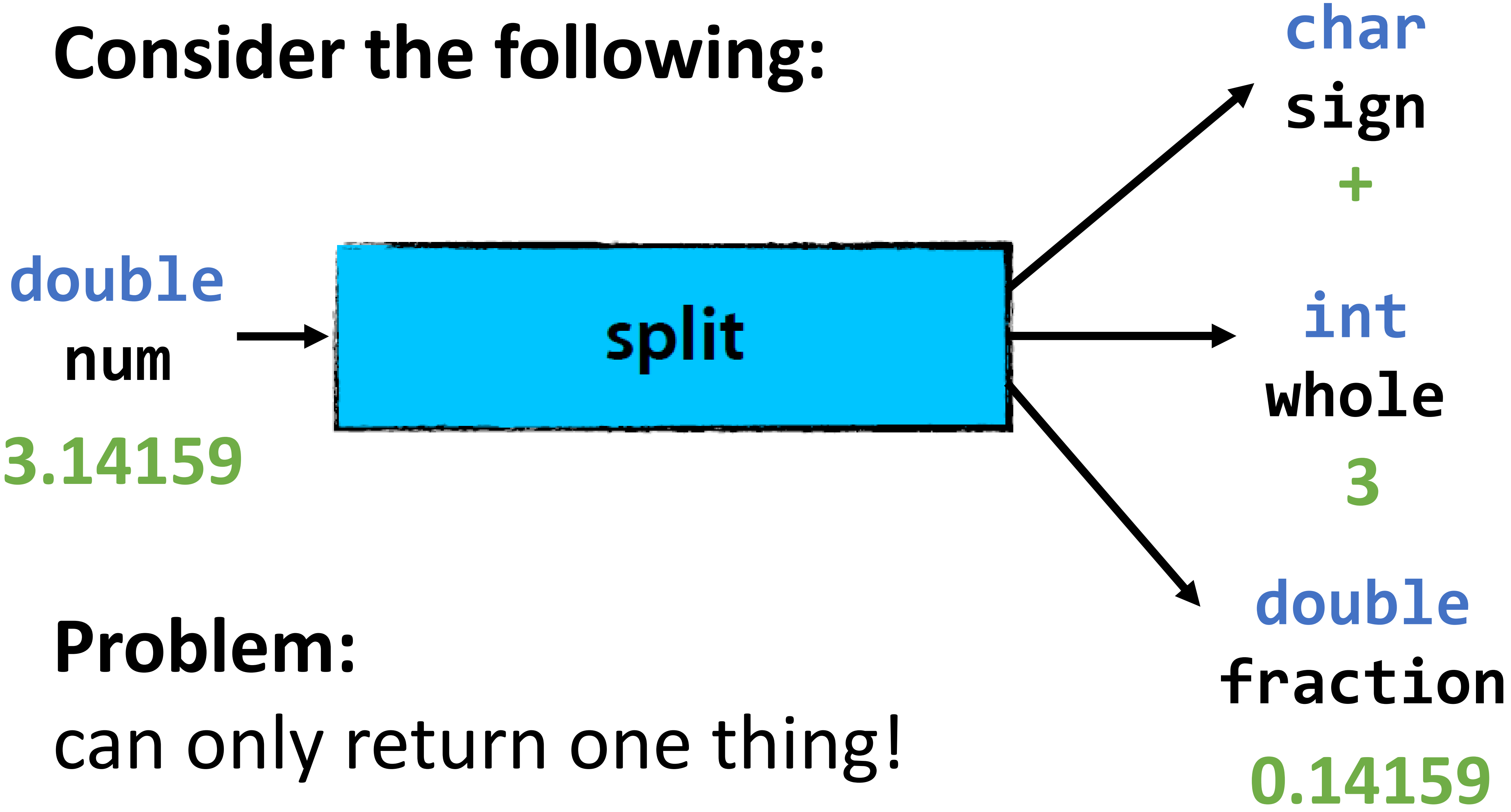
Multiple results?

(WITHOUT using global variables!)



POINTERS to the rescue!

Consider the following:



double
num
3.14159

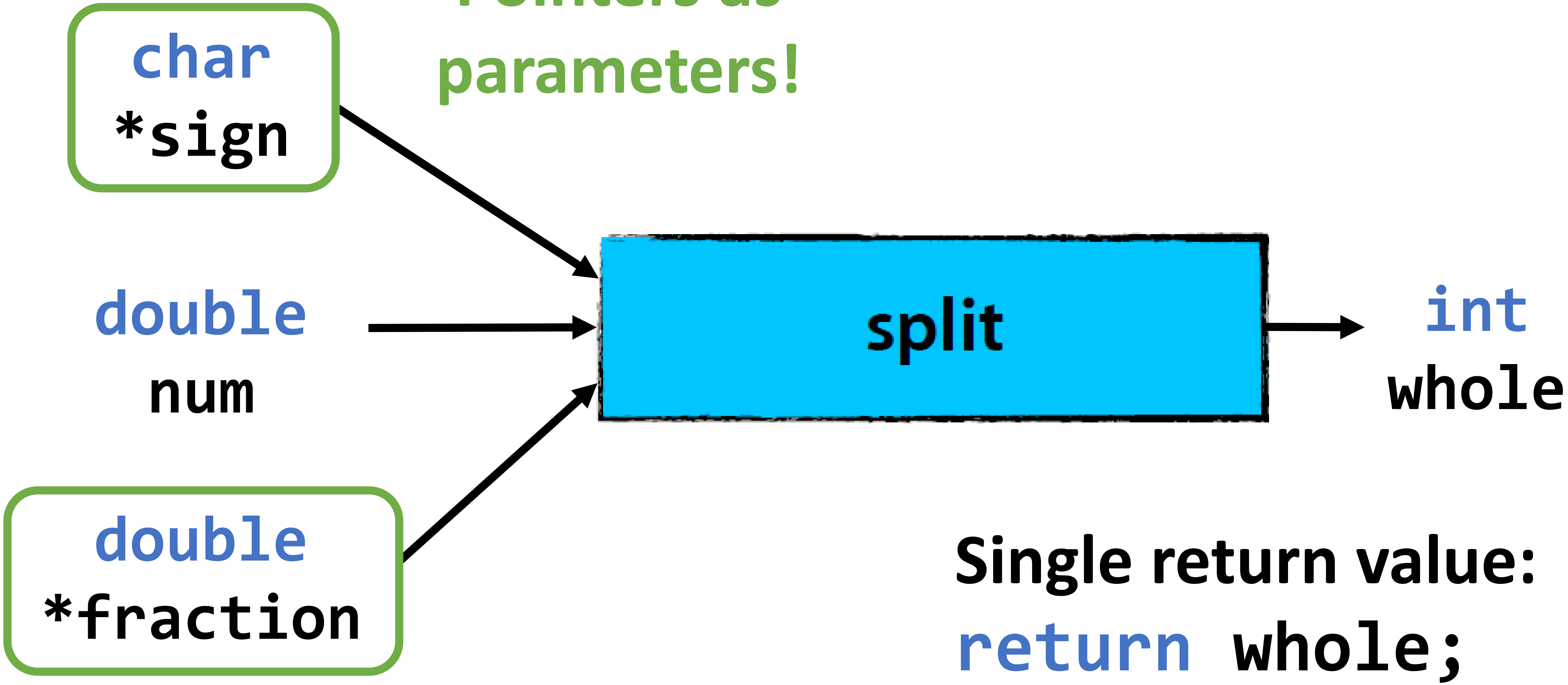
split

char
sign
+

int
whole
3

double
fraction
0.14159

Problem:
can only return one thing!



Pointers as parameters!

char *sign

double num

double *fraction

split

int whole

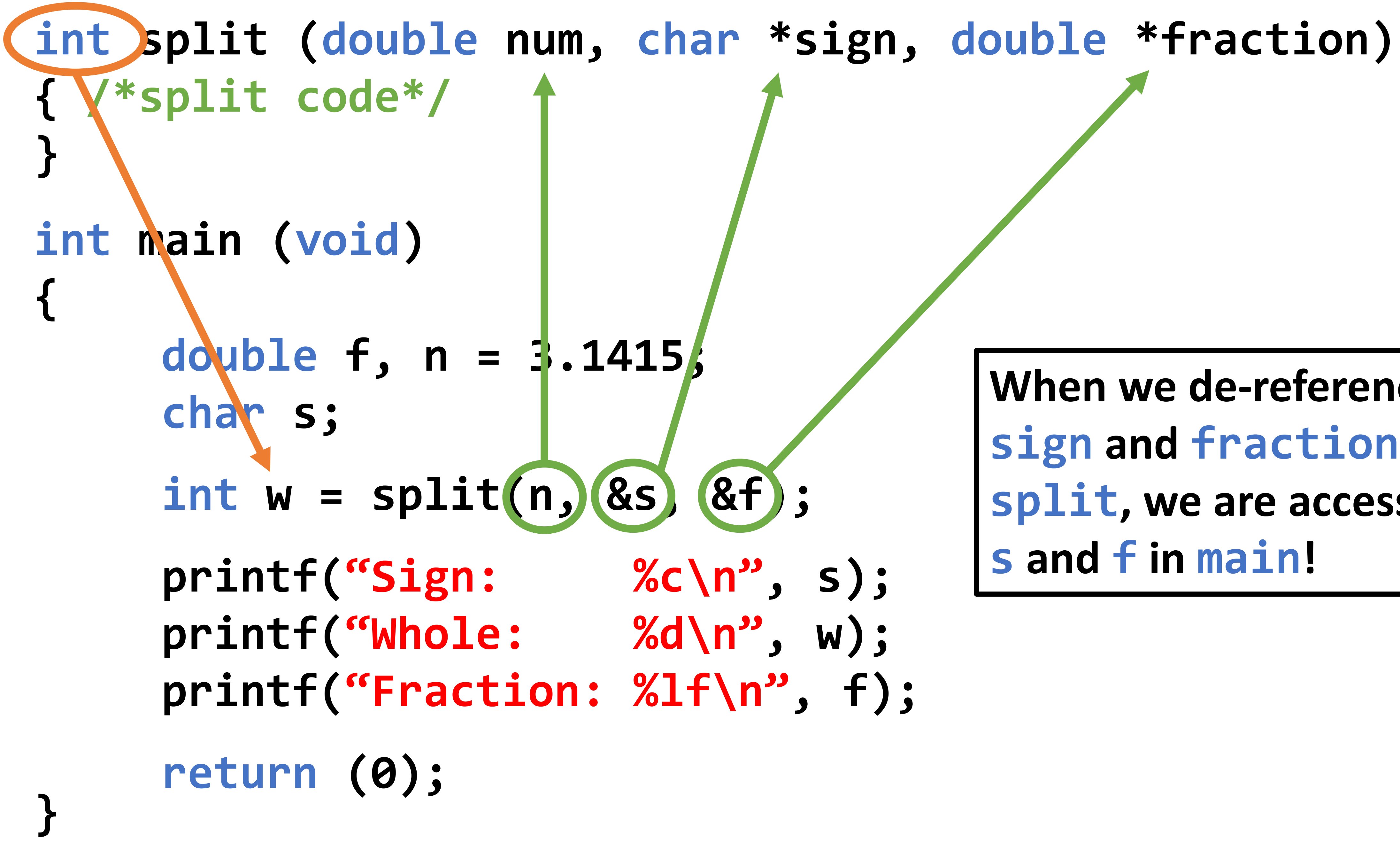
Single return value:

```
return whole;
```

```
int split (double num, char *sign, double *fraction)
{
    int whole = abs((int)num);
    *fraction = fabs(num) – whole;
    if (num >= 0)
        *sign = ‘+’;
    else
        *sign = ‘-’;
    return (whole);
}
```

Pointers

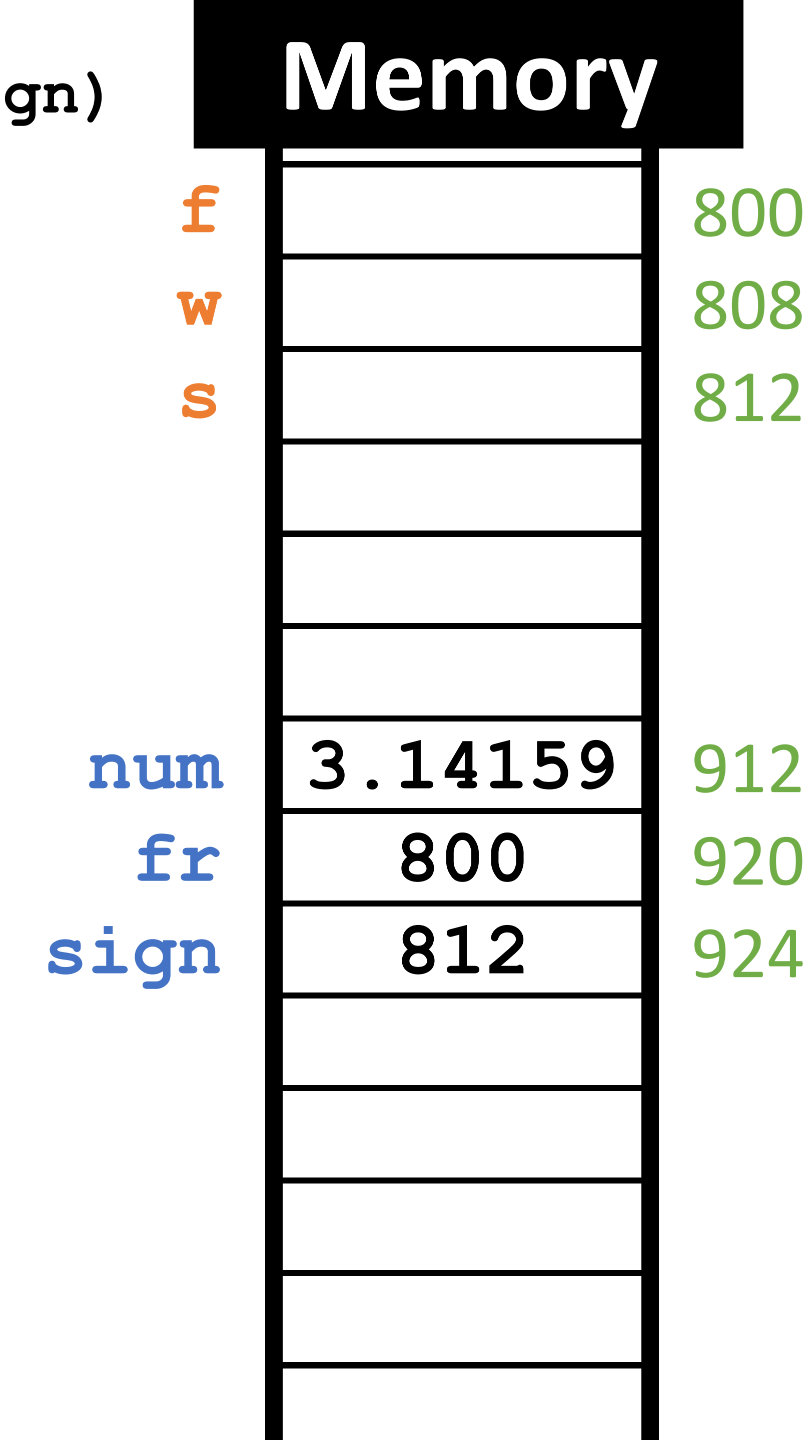
De-reference pointers to assign values



```
int split (double num, char *sign, double *fraction)
{ /*split code*/
}

int main (void)
{
    double f, n = 3.1415;
    char s;
    int w = split(n, &s, &f);
    printf(“Sign:     %c\n”, s);
    printf(“Whole:    %d\n”, w);
    printf(“Fraction: %lf\n”, f);
    return (0);
}
```

When we de-reference sign and fraction in split, we are accessing s and f in main!

```
int split (double num, double *fr, char *sign)
{
     int whole = abs((int)num);
     *fr = fabs(num) - whole;
     if (num >= 0)   *sign = ‘+’;
     else            *sign = ‘-’;
     return (whole);
}

int main (void)
{
     double f; int w; char s;
     w = split(3.14159, &f, &s);
     printf(“%c %d %lf”, s, w, f);
     return (0);
}
```



Memory

f		800
w		808
s		812
num	3.14159	912
fr	800	920
sign	812	924

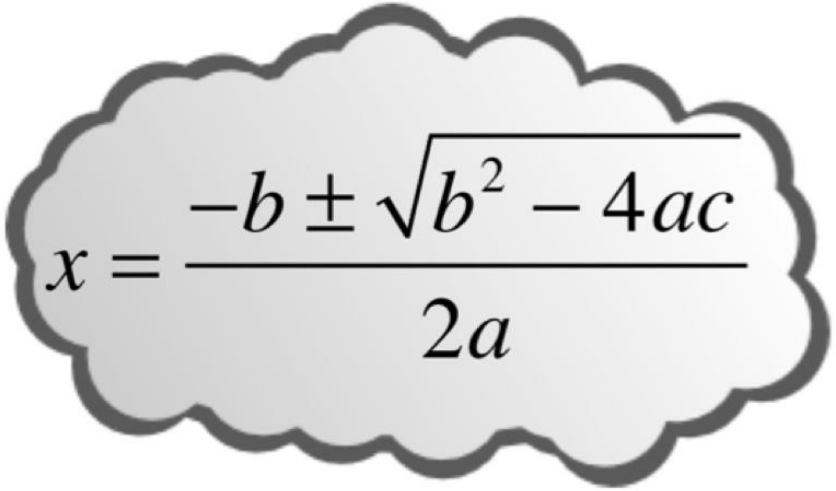
```c
int split (double num, double *fr, char *sign)
{
    int whole = abs((int)num);  /* 3 */
    *fr = fabs(num) - whole; /* 0.14159 */
    if (num >= 0)   *sign = '+';
    else   *sign = '-';
    return (whole);
}

int main (void)
{
    double f; int w; char s;
    w = split(3.14159, &f, &s);
    printf("%c %d %lf", s, w, f);
    return (0);
}
```

Memory

f	0.14159	800
w	3	808
s	'+'	812
num	3.14159	912
fr	800	920
sign	812	924
whole	3	928

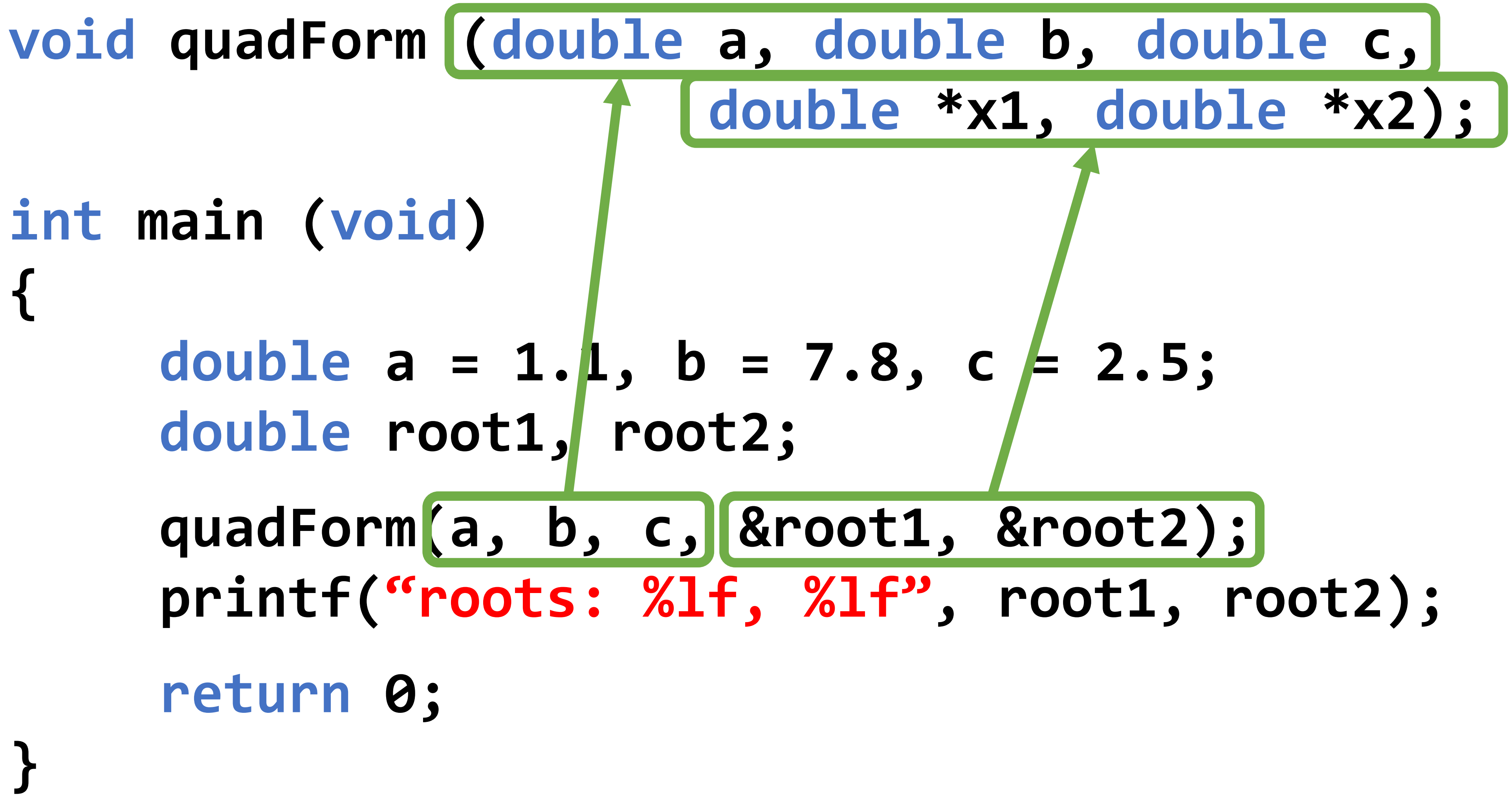
Quadratic Formula

```
void quadForm (double a, double b, double c,
                                double *x1, double *x2)
{
      double tmp = sqrt(b*b – 4*a*c);

      *x1 = (-b + tmp)/(2*a);
      *x2 = (-b - tmp)/(2*a);
}
```

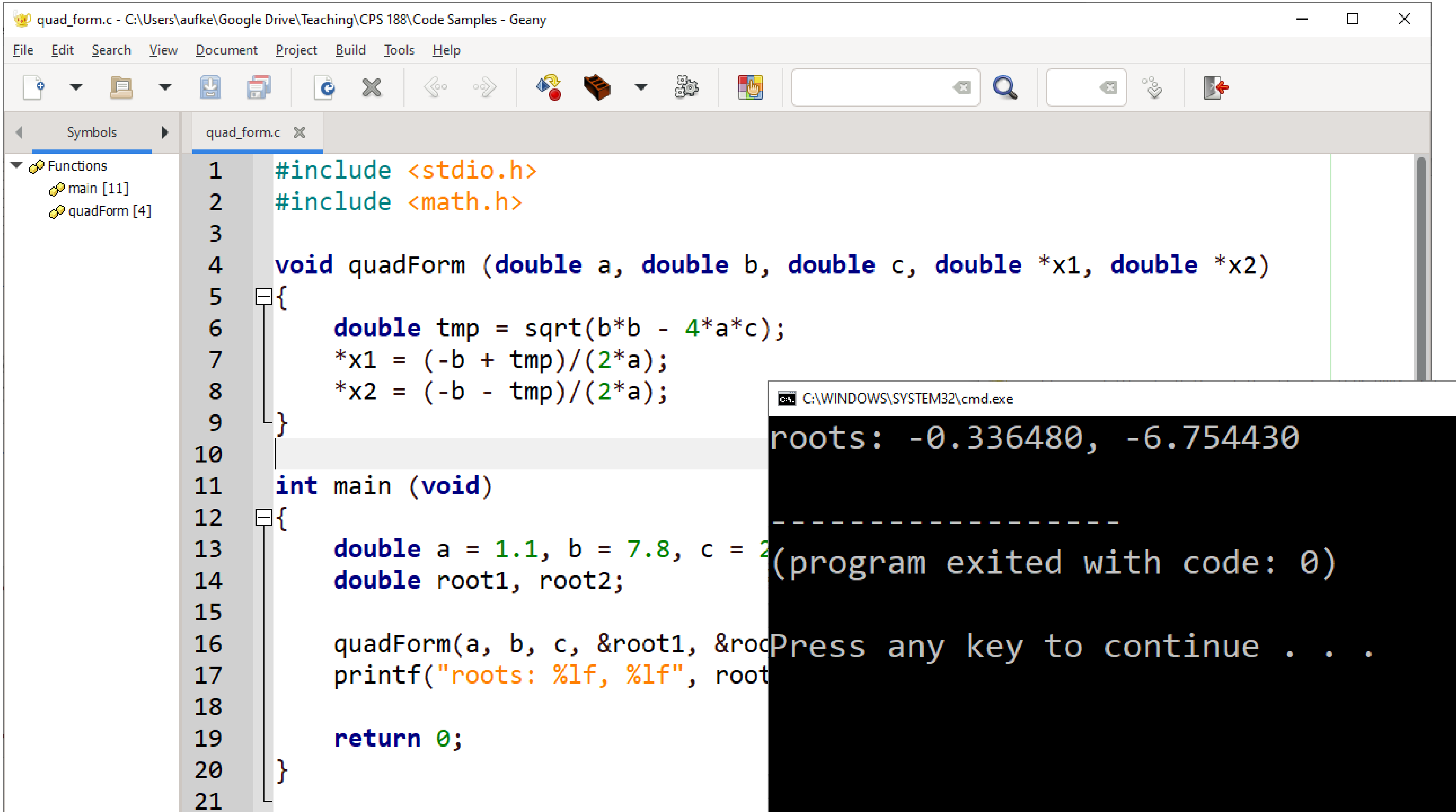


$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Quadratic Formula



```
void quadForm (double a, double b, double c,
                      double *x1, double *x2);

int main (void)
{
    double a = 1.1, b = 7.8, c = 2.5;
    double root1, root2;

    quadForm(a, b, c, &root1, &root2);
    printf(“roots: %lf, %lf”, root1, root2);

    return 0;
}
```


quad_form.c - C:\Users\aufke\Google Drive\Teaching\CPS 188\Code Samples - Geany
File
Edit
Search
View
Document
Project
Build
Tools
Help
Symbols
quad_form.c
Functions
main [11]
quadForm [4]
C:\WINDOWS\SYSTEM32\cmd.exe

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

void quadForm (double a, double b, double c, double *x1, double *x2)
{
    double tmp = sqrt(b*b - 4*a*c);
    *x1 = (-b + tmp)/(2*a);
    *x2 = (-b - tmp)/(2*a);
}

int main (void)
{
    double a = 1.1, b = 7.8, c = 2
    double root1, root2;

    quadForm(a, b, c, &root1, &roc
    printf("roots: %lf, %lf", root

    return 0;
}
```

```
roots: -0.336480, -6.754430

------------------
(program exited with code: 0)

Press any key to continue . . .
```

Questions?