CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 3.2: More functions, variable scope, globals

Toronto Metropolitan University

Notice!

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

Today

- Function examples
- More functions: stdlib.h
- Function scope, global variables

Example: Circle Calculator

Our first Python program that does something meaningful:

Task:

- Write a C program that asks the user to enter a value for the radius of a circle.
- Your program should use the radius to calculate and print the circumference and the area of the circle.

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

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

double circumference (double r)
{
   return 2*PI*r;
}
```

- We already wrote an area function last class.
- Let's just reuse it!
- We can also write a circumference() function.

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

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

double circumference (double r)
{
   return 2*PI*r;
}
```

- We already wrote an area function last class.
- Let's just reuse it!
- We can also write a circumference() function.

```c
int main (void)
{
   double r;
   printf("Enter the radius: ");
   scanf("%lf", &r);
   printf("Area: %.2lf\n", circ_area(r));
   printf("Circumference: %.2lf\n" , circumference(r));
   return 0;
}
```

Symbols

- Functions
  - circ_area [4]
  - circumference [9]
  - main [14]
- Macros
  - PI [2]

circ_calc.c

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

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

double circumference (double r)
{
    return 2*PI*r;
}

int main (void)
{
    double r;
    printf("Enter the radius: ");
    scanf("%lf", &r);
    printf("Area: %.2lf\n", circ_area(r));
    printf("Circumference: %.2lf\n" , circumference(r));
    return 0;
}
```

C:\WINDOWS\SYSTEM32\cmd.exe

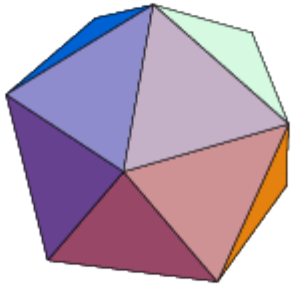
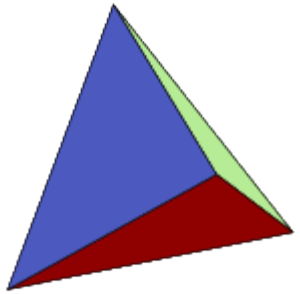
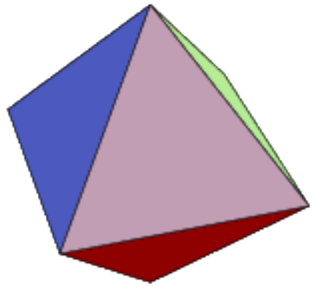
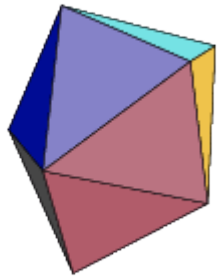
```
Enter the radius: 5
Area: 78.54
Circumference: 31.42


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

Press any key to continue . . .
```

Example: Deltahedron Calculator

<u>Write a program to do the following:</u>

Compute the surface area of a deltahedron, given the number of faces and the edge length. Use multiple functions.

A deltahedron is a solid whose faces are all equilateral triangles

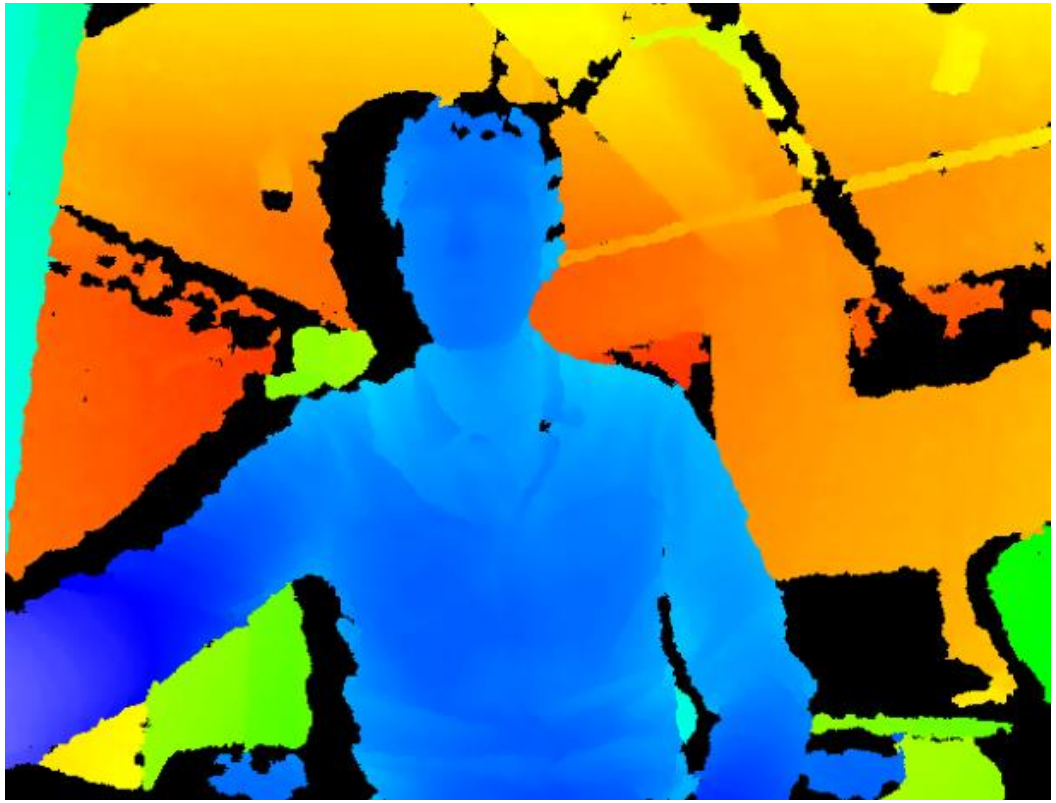
To `main()`

From `main()`

```
double deltahedron_area(double len, int nFaces)
{
        double surfaceArea;

        surfaceArea = nFaces * equTriArea(len);

        return surfaceArea;
}

double equTriArea(double sideLength)
{
        double a;

        a = (sqrt(3)/4)*sideLength*sideLength;

        return a;
}
```

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

double deltahedron_area(double len, int nFaces);
double equTriArea(double sideLength);

int main (void)
{
      int n;
      double length, sa;

      printf("Enter number of faces, length of sides");
      scanf("%d %lf", &n, &length);

      sa = deltahedron_area(length, n);

      printf("Surface area is %.2lf\n", sa);

      return 0;
}
```

Bitwise Example

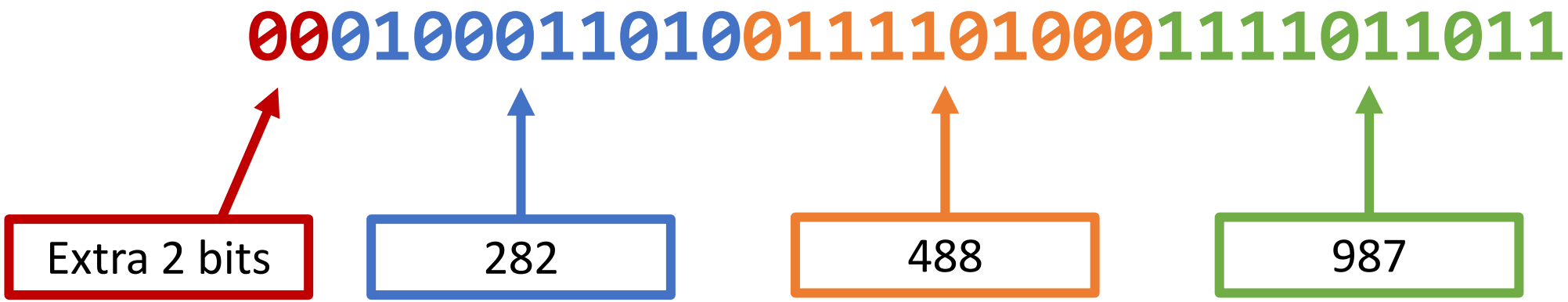


- Kinect depth values: 10-bit resolution
- I.e., no value larger than 1024
- Could use an **`int`**, but that's 32 bits.
- Waste! Better to use a **`short`**.
- 16 bits, still wasting 6 bits.
- If we know how to mask and manipulate bits, we can pack three numbers into a 32-bit integer.
- Now, only two bits are wasted every three numbers

Example: Integer Packing

Shift 10 bits at a time:

- Consider three numbers: **282**, **488**, **987**
- We want to pack them into a single **`int`** like so:



intpack.c - C:\Users\aufke\Google Drive\Teaching\CPS 188\Code Samples - Geany

File Edit Search View Document Project Build Tools Help

Symbols

- Functions
  - main [17]
  - pack3 [3]
  - unpack3 [10]

intpack.c

```c
#include <stdio.h>

int pack3 (int a, int b, int c)
{
    int packed = a << 10;
    packed = (packed | b) << 10;
    return packed | c;
}

void unpack3 (int packed)
{
     printf("%d\n", packed & 0x3FF);
     printf("%d\n", (packed >> 10) & 0x3FF);
     printf("%d\n", (packed >> 20) & 0x3FF);
}

int main (void)
{
    int packed = pack3(282, 488, 987);
    printf("packed = %d\n", packed);
    unpack3(packed);
    return 0;
}
```

C:\WINDOWS\SYSTEM32\cmd.exe

```
packed = 296199131
987
488
282


------------------
(program exited with code:
```

Variable SCOPE

Variables declared inside a function are ***unknown*** outside of that function

```
int my_increment (int i)
{
    int j;

    j = i + 1;
    return j;
}
```

Variable scope

Problem?

```
int main(void)
{
    int k = 1;
    printf("k+1: %d\n", my_increment(k) );
    printf("j: %d\n", j );
    return (0);
}
```

Using j outside of its scope! Bad!

Problem?

```
int my_increment (int i)
{
    int j;

    j = k + 1;
    return j;
}

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

k not declared in this scope!

Scope of variable k

```c
#include <stdio.h>

int inc(int i, int j)
{
    i++; j++;
    return (i + j);
}

int main (void)
{
    int i = 1, j = 5;
    printf("i+j = %d \n", inc(i, j));
    printf("i = %d, j = %d \n", i, j);
    return (0);
}
```

Console

```
i + j = 8
i = 1, j = 5
```

```c
#include <stdio.h>
int inc(int i, int j)
{
    i++; j++;
    return (i + j);
}
int main (void)
{
    int i = 1, j = 5;
    printf("i+j = %d \n", inc(i, j));
    printf("i = %d, j = %d \n", i, j);
    return (0);
}
```

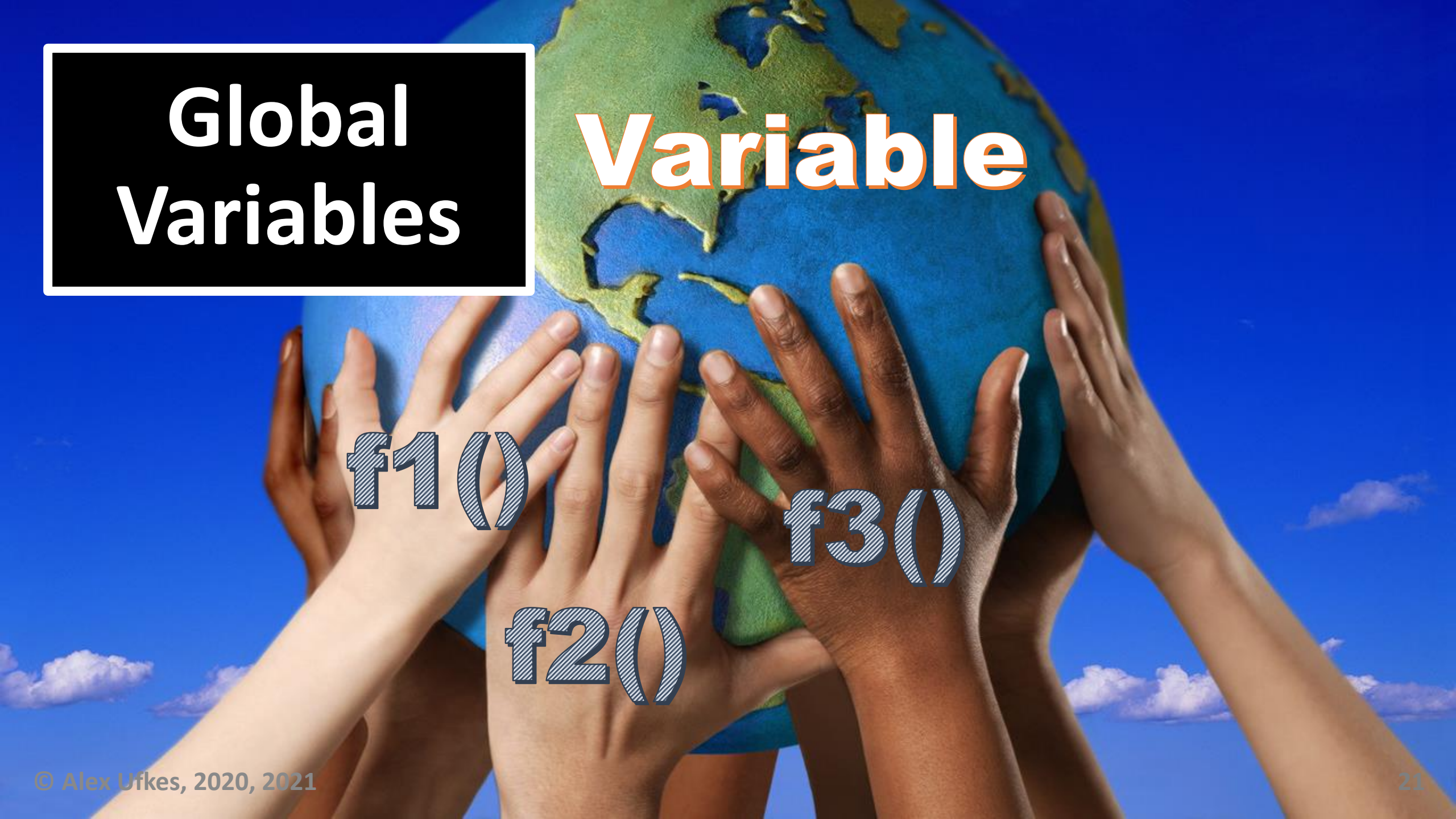
Changes in this scope affect the function parameters **i** and **j**.

These are not affected!

```c
#include <stdio.h>
int inc(int i, int j)
{
    i++; j++;
    return (i + j);
}
int main (void)
{
    int i = 1, j = 5;
    printf("i+j = %d \n", inc(i, j));
    printf("i = %d, j = %d \n", i, j);
    return (0);
}
```

Variables can have the same name, so long as they do not have the same ***scope***.

Global Variables



```c
#include <stdio.h>

double xyz = 15.0;  /* Global variable */

void f1 (void)
{
     printf("%.2lf\n", xyz);
}

int main (void)
{
     f1();
     printf("%.2lf\n", xyz);
     return 0;
}
```

Global variables are declared outside of any function.

Output

```
15.00
15.00
```

Scope of xyz

What about this?

```
#include <stdio.h>

void f1 (void)
{
     printf("%.2lf\n", xyz);
}

double xyz = 15.0;

int main (void)
{
     f1();
     printf("%.2lf\n", xyz);
     return 0;
}
```

COMPILE ERROR!

`xyz` not in scope of **`f1()`**.

Scope of xyz

Is this Allowed?

```
#include <stdio.h>

int xyz = 15;

int main (void)
{
    int xyz = 17;
    printf("%d\n", xyz);
    return 0;
}
```

Two variables, same name!

What about their scope?

Same name, different scope = allowed

Is this Allowed?

```
#include <stdio.h>

int xyz = 15;

int main (void)
{
    int xyz = 17;
    printf("%d\n", xyz);
    return 0;
}
```

Which xyz is printed?

Output

17

Whichever is closest in scope.

Global Variable Caveat

You may be tempted to use global variables for **everything**,
and just avoid scope issues entirely

DON’T.

This goes against principles of modular programming.
We want our functions to be as self-contained as possible.

Global Variable Caveat

Meaning, the **scope** of our variables should be as **small** as possible.

When should they be used?

There's no set rule, but if there is a single variable being used by multiple functions, a global variable could be used

stdlib.h

- This libaray contains many other useful functions
- Probably the 3rd most common, after stdio.h and math.h
- A selection of functions:

```
abs(a);          // Returns absolute value of an integer
                 // (fabs(a) in math.h is for doubles)
n = rand();      // Random integer between 0 and RAND_MAX
```

RAND_MAX?

rand()

dice_roll.c

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

int main (void)
{
    printf("RAND_MAX = %d\n", RAND_MAX);
    return 0;
}
```

C:\WINDOWS\SYSTEM32\cmd.exe

```
RAND_MAX = 32767


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

Press any key to continue . . .
```

32767 = Maximum value of a signed 16-bit integer!

rand()

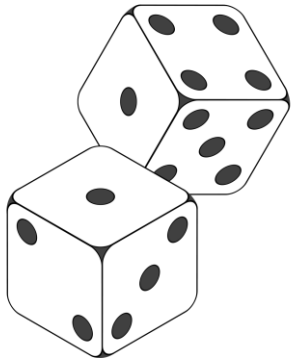
dice_roll.c

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

int main (void)
{
    printf("RAND_MAX = %d\n", RAND_MAX);
    printf("%d\n", rand());
    printf("%d\n", rand());
    printf("%d\n", rand());
    printf("%d\n", rand());
    printf("%d\n", rand());
    return 0;
}

```

C:\WINDOWS\SYSTEM32\cmd.exe

```
RAND_MAX = 32767
41
18467
6334
26500
19169


------------------
```

Dice Roll



- rand() is between 0-RAND_MAX
- What if I want to roll a dice? We need a random value between 1-6
- How can this be done?

```
int dice = (rand() % 6) + 1;
```

Between 0-5

Between 1-6

dice_roll.c

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

int main (void)
{
    printf("RAND_MAX = %d\n", RAND_MAX);
    printf("%d\n", rand() % 6 + 1);
    printf("%d\n", rand() % 6 + 1);
    printf("%d\n", rand() % 6 + 1);
    printf("%d\n", rand() % 6 + 1);
    printf("%d\n", rand() % 6 + 1);
    return 0;
}
```

C:\WINDOWS\SYSTEM32\cmd.exe

```
RAND_MAX = 32767
6
6
5
5
6


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

Press any key to continue . . .
```

- If you run this again and again, you'll get the same results every time
- Not so random! What gives?

`srand()`

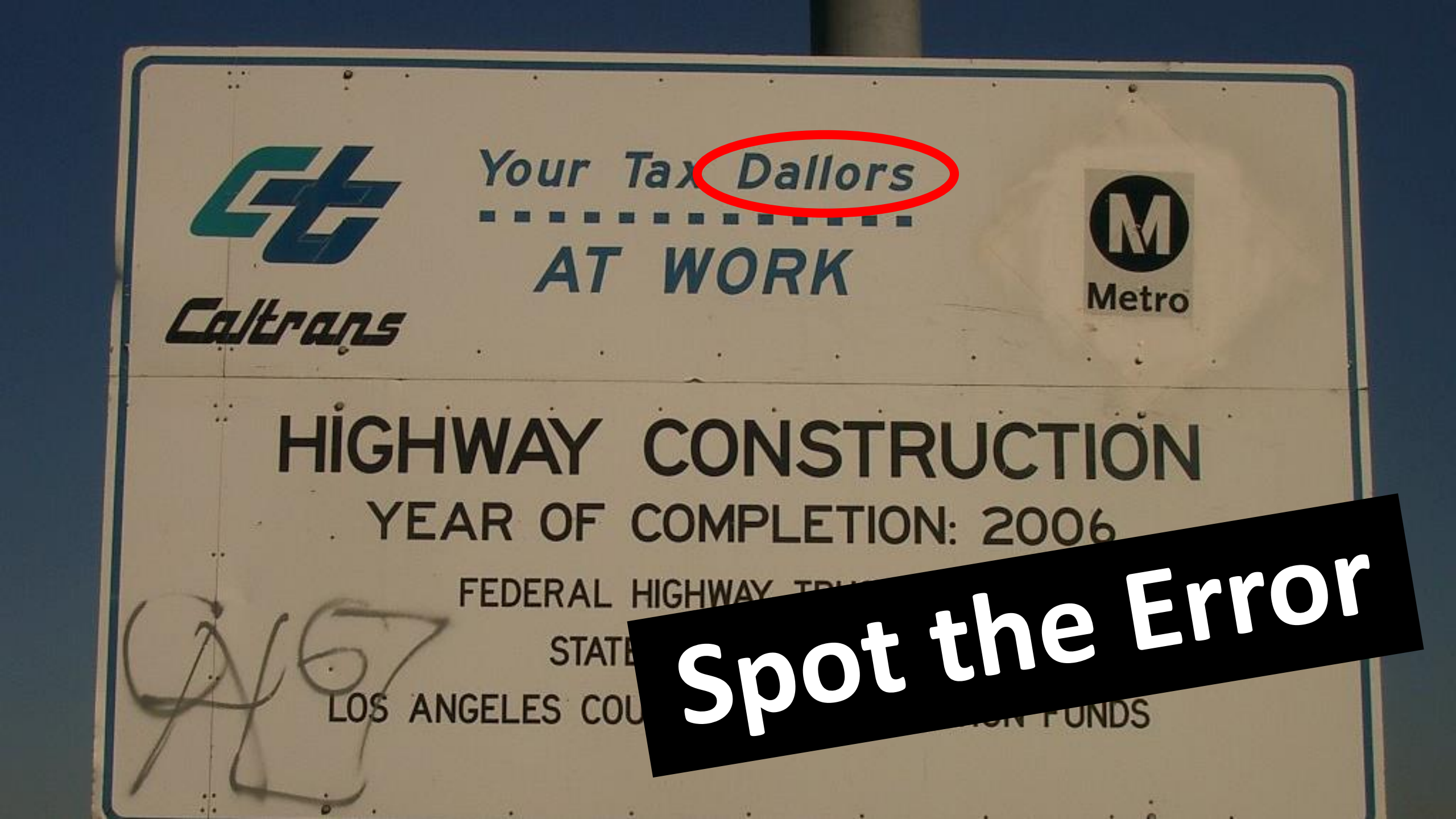
- There's no such thing as "random" in a computer, not truly.
- Random numbers are produced using a sophisticated function whose next value depends on the previous value.
- This equation must be ***seeded*** with an initial value
- Same initial seed? Same sequence of random numbers.
- We can use the **`srand()`** function to set this initial seed.

dice_roll.c

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

int main (void)
{
    srand(42);
    printf("%d\n", rand() % 6 + 1);
    printf("%d\n", rand() % 6 + 1);
    printf("%d\n", rand() % 6 + 1);
    printf("%d\n", rand() % 6 + 1);
    printf("%d\n", rand() % 6 + 1);
    return 0;
}
```

C:\WINDOWS\SYSTEM32\cmd.exe

```
2
5
2
3
4


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

Press any key to continue . . .
```


Caltrans
Your Tax Dallors
AT WORK
Metro
HIGHWAY CONSTRUCTION
YEAR OF COMPLETION: 2006
FEDERAL HIGHWAY
STATE
LOS ANGELES COU
FUNDS
Spot the Error

```
#include <stdio.h>

void f1 (int x)
{
    return x + 5
}

int main (void)
{
    int y = 15;
    printf("%d\n", f1(y + 2));
    return 0;
}
```

void function cannot return a value

```
#include <stdio.h>

int f1 (int x)
{
    int q = x + 5;
    return q;
}

int main (void)
{
    int y = 15;
    printf("%d\n", f1(y + 2));
    printf("%d\n", q);
    return 0;
}
```

main is outside the scope of q

```
#include <stdio.h>
void change (int newVal, int num)
{
     num = newVal;
}

int main (void)
{
     int number = 17;

     printf("Original: %d", number);
     change(14, 12, 13);
     printf("Changed: %d", number);

     return 0;
}
```

Parameter/argument count mismatch

In Summary

Functions

Return values, arguments & parameters, function scope, global variables

Questions?