

CPS 188

Computer Programming Fundamentals

Prof. Alex Ufkes

Topic 3.2: More functions, variable scope, globals

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.

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.

```
#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.

```
#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;
}
```

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

```

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  double circumference (double r)
10 {
11     return 2*PI*r;
12 }
13
14 int main (void)
15 {
16     double r;
17     printf("Enter the radius: ");
18     scanf("%lf", &r);
19     printf("Area: %.2lf\n", circ_area(r));
20     printf("Circumference: %.2lf\n", circumference(r));
21     return 0;
22 }
23
    
```

```

C:\WINDOWS\SYSTEM32\cmd.exe

Enter the radius: 5
Area: 78.54
Circumference: 31.42

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

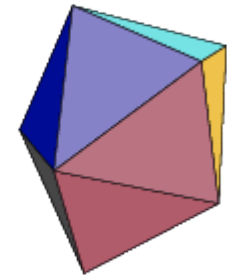
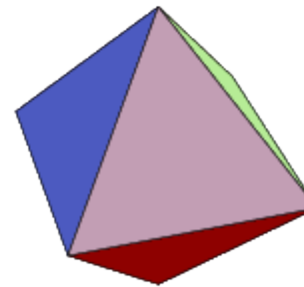
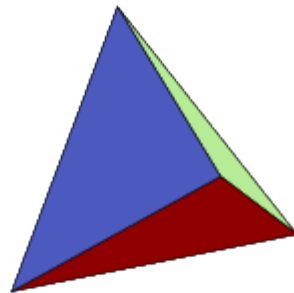
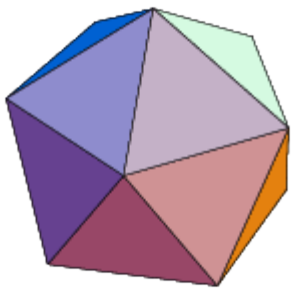
Press any key to continue . . .
    
```

Example: Deltahedron Calculator

Write a program to do the following:

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;
}
```

```
graph TD
    subgraph ToMain [To main()]
        RA[return surfaceArea]
    end
    subgraph FromMain [From main()]
        C[equTriArea(len)]
        R2[return a]
    end
    RA --> TM[To main()]
    C --> FM[From main()]
    R2 --> FM
```

```
#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;

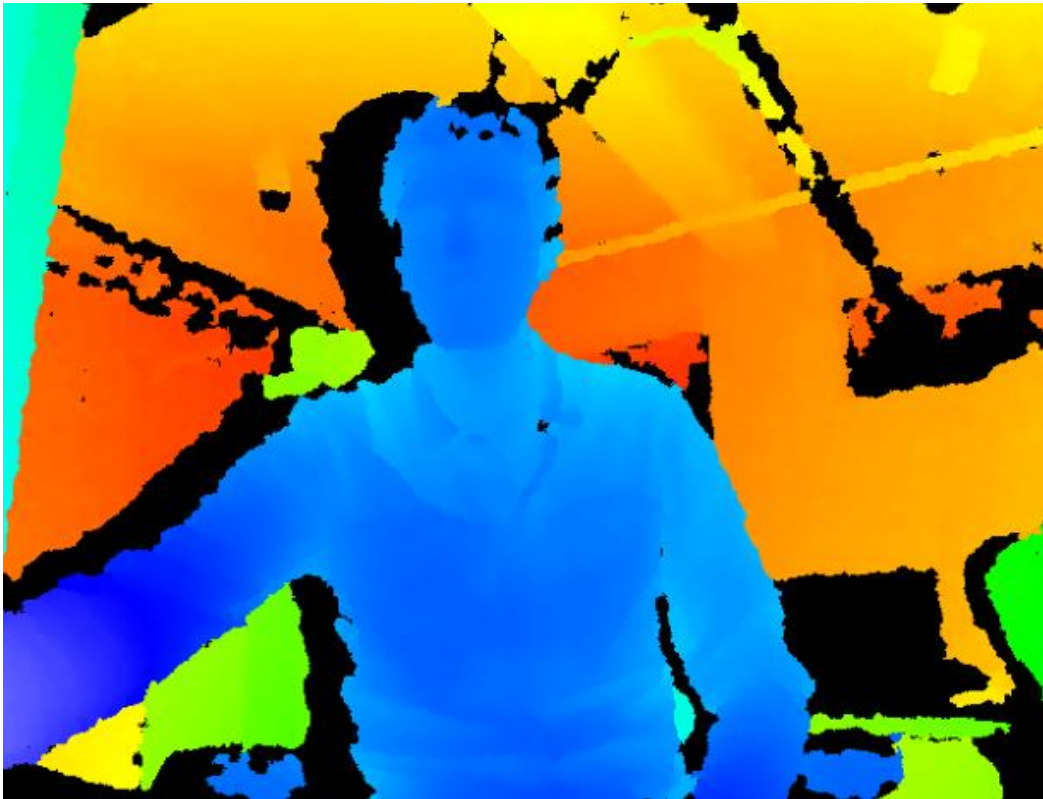
    printf("Enter number of faces, length of sides");
    scanf("%d %lf", &n, &length);

    sa = deltahedron_area(length, n);

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

    return 0;
}
```

Bitwise Example

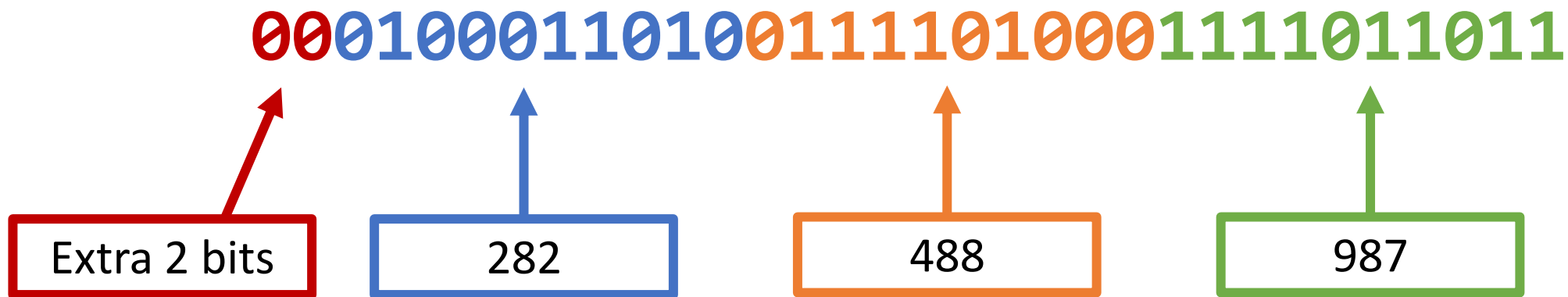


- Kinect depth values: 10-bit resolution
- I.e., no value larger than 1024
- Could use an **int**, but that's 32 bits.
- Waste! Better to use a **short**.
- 16 bits, still wasting 6 bits.
- If we know how to mask and manipulate bits, we can pack three numbers into a 32-bit integer.
- Now, only two bits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





```
1  #include <stdio.h>
2
3  int pack3 (int a, int b, int c)
4  {
5      int packed = a << 10;
6      packed = (packed | b) << 10;
7      return packed | c;
8  }
9
10 void unpack3 (int packed)
11 {
12     printf("%d\n", packed & 0x3FF);
13     printf("%d\n", (packed >> 10) & 0x3FF);
14     printf("%d\n", (packed >> 20) & 0x3FF);
15 }
16
17 int main (void)
18 {
19     int packed = pack3(282, 488, 987);
20     printf("packed = %d\n", packed);
21     unpack3(packed);
22     return 0;
23 }
```

C:\WINDOWS\SYSTEM32\cmd.exe

packed = 296199131

987

488

282

(program exited with code:

A landscape view through a telescope scope. The scope's circular field of view is centered on a body of water, with a grassy hill in the background and a dry, grassy field in the foreground. The text "Variable SCOPE" is overlaid on the center of the scope's view. The word "Variable" is in white, and "SCOPE" is in blue with a white outline. The background landscape features a clear blue sky, a body of water, a grassy hill, and a dry, grassy field with a fence line in the foreground.

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;
}
```

**k not declared in
this scope!**

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

Scope of variable k

```
#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

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

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

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

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

```
int i = 1, j = 5;
```

These are not affected!

```
printf("i+j = %d \n", inc(i, j));  
printf("i = %d, j = %d \n", i, j);  
return (0);
```

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

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

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

```
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);
}
```

Global Variables

Variable

`f1()`

`f3()`

`f2()`

```
#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);
```

```
}
```

COMPILE ERROR!

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

```
double xyz = 15.0;
```

```
int main (void)
```

```
{
```

```
    f1();
```

```
    printf("%.2lf\n", xyz);
```

```
    return 0;
```

```
}
```

Scope of xyz

Is this Allowed?

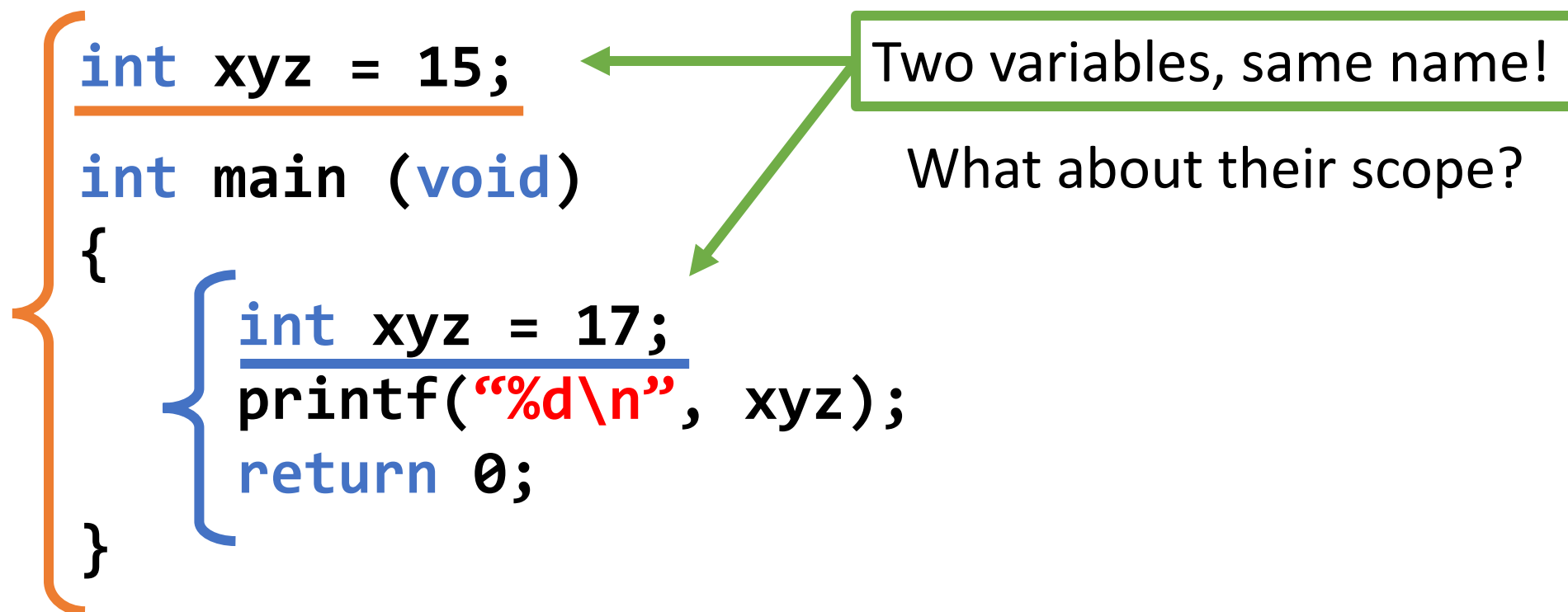
```
#include <stdio.h>

int xyz = 15;

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

Two variables, same name!

What about their scope?

The diagram uses curly braces to delineate the scope of the variables. An orange brace on the left groups the global variable 'int xyz = 15;' and the entire 'main' function. A blue brace on the right groups the local variable 'int xyz = 17;' and the two lines of code it follows ('printf' and 'return'). Two green arrows originate from a green-bordered box containing the text 'Two variables, same name!'. One arrow points to the global variable 'int xyz = 15;', and the other points to the local variable 'int xyz = 17;'. Below the box, the text 'What about their scope?' is written.

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 library 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 x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main (void)
5  {
6      printf("RAND_MAX = %d\n", RAND_MAX);
7      return 0;
8  }
9
10
```

```
C:\WINDOWS\SYSTEM32\cmd.exe
RAND_MAX = 32767
-----
(program exited with code: 0)
Press any key to continue . . .
```

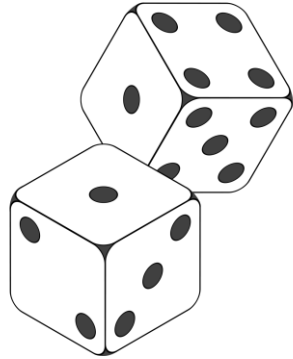
32767 = Maximum value of
a signed 16-bit integer!

rand()

```
dice_roll.c ✕
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main (void)
5  {
6      printf("RAND_MAX = %d\n", RAND_MAX);
7      printf("%d\n", rand());
8      printf("%d\n", rand());
9      printf("%d\n", rand());
10     printf("%d\n", rand());
11     printf("%d\n", rand());
12     return 0;
13 }
14
```

```
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 x
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main (void)
5  {
6      printf("RAND_MAX = %d\n", RAND_MAX);
7      printf("%d\n", rand() % 6 + 1);
8      printf("%d\n", rand() % 6 + 1);
9      printf("%d\n", rand() % 6 + 1);
10     printf("%d\n", rand() % 6 + 1);
11     printf("%d\n", rand() % 6 + 1);
12     return 0;
13 }
14
15
```

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

```
C:\WINDOWS\SYSTEM32\cmd.exe
RAND_MAX = 32767
6
6
5
5
6

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

Press any key to continue . . .
```


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 ✕
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main (void)
5  {
6      srand(42);
7      printf("%d\n", rand() % 6 + 1);
8      printf("%d\n", rand() % 6 + 1);
9      printf("%d\n", rand() % 6 + 1);
10     printf("%d\n", rand() % 6 + 1);
11     printf("%d\n", rand() % 6 + 1);
12     return 0;
13 }
14
15
```

```
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



HIGHWAY CONSTRUCTION

YEAR OF COMPLETION: 2006

FEDERAL HIGHWAY TRUST

STATE

LOS ANGELES COUNTY

CONSTRUCTION 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?

