CSE 220 – C Programming

Functions

Exam

- 11 Nov, 50 minutes
- Mimir, similar with an assignment, but no visible test cases
- Screen share and keep camera open
- Release the exam samples on Friday this week, review them on 9 Nov.

Mid-semester Evaluation

- Comments and Suggestions
  - https://www.egr.msu.edu/mid-semester-evaluation

Functions

- Building blocks of C programs
- Divide program into smaller pieces
- Easier to understand
- Easier to maintain
- Reuse code and avoid repetition

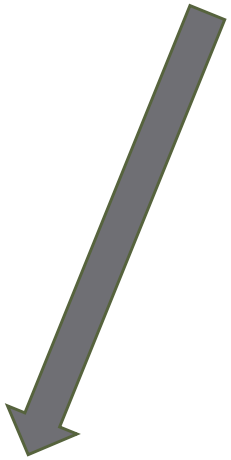
Example

```
int main(void) {
   printf(“*********\n”);
   printf(“Select one of the following options\n:”);
   printf(“- A: to convert from oz to lbs\n”);
   printf(“- B: to convert from lbs to oz\n”);
   printf(“- C: to convert from g to lbs\n”);
   printf(“- D: to convert from lbs to g\n”);
   printf(“*********\n”);
   char option;
   scanf(“%c”, &option);
   //Ask for number and apply the conversion
accordingly
   ...}
```

Example

```
int main(void) {
    char option;
    displayOptions();
    scanf("%c", &option);
    //Ask for number
    …
    //Apply conversion
    float result;
    switch(option) {
        case 'A':   result = convertOzToLbs(number);
                    break;
        …
```

- Simplify Code
- Must be defined in the code

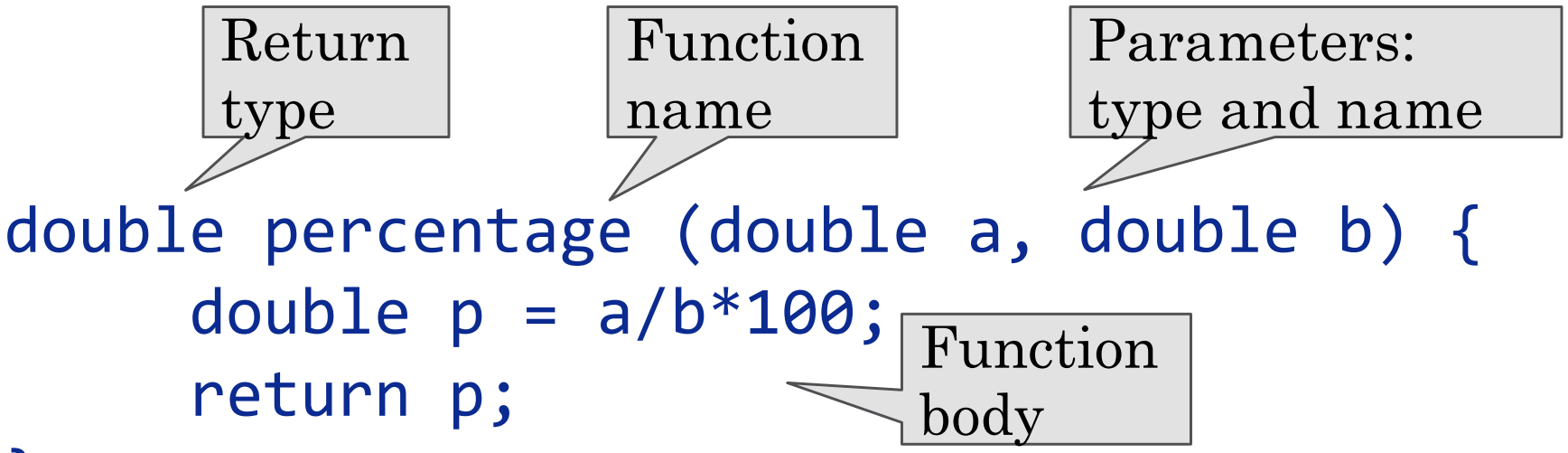


Defining a function

```
double percentage(double a, double b) {
  double p = a/b*100;
  return p;
}
```

- Must specify:
  - Return type: double
  - function name: percentage
  - function parameters and type: double a, double b
  - function body

Defining a function



Exercise

What is the return type for this function definition?

```
char get_letter(int x, char word[]) {
  return word[x];
}
```

- `char`
- `int`
- `char[]`
- `void`

Defining a function

- Functions don't have to return anything: Return type is void
- Functions cannot return arrays
- Functions don't have to take input parameters: use void in place of parameters

```
void sayHello(void) {
    printf("Hello everyone\n");
}
```

Defining a function

- The type must be listed for every parameter

```
void add(int x, y) {
    printf("%d + %d = %d\n", x,
           y, x+y);
}
```

WRONG

```
void add(int x, int y) {
    printf("%d + %d = %d\n", x,
           y, x+y);
}
```

CORRECT

Standard Library Functions

math.h

```
double pow(
       double a, double b);
double sqrt(double a);
double ceil(double a);
double floor(double a);
double log(double a);
double log10(double a);
double exp(double a);
double cos(double a);
double sin(double a);
double tan(double a);
```

stdlib.h

```
int abs(int x);
int rand(void);
/* returns int
between 0 and RAND_MAX (a
very large number)
*/

void exit(int status);
void abort(void);
int system(
       const char *string);
```

Calling a function

- Function call, the function's name plus the arguments in parentheses.

```
z = average(x, y);

average(x, y);     //don’t capture result

sayHello();
```

- The parenthesis are required, even if there are no arguments to provide.

```
sayHello;      //Wrong
```

Example

```
#include <stdio.h>
double percentage (double a, double b) {
  double p = a/b*100;
  return p;
}
int main (void) {
  double x = 5, y = 20;
  double val = percentage(x, y);
  printf("%f to %f is %f %%\n", x, y, val);
  printf("7 to 35 is %f %%\n", percentage(7, 35));
  return 0;
}
```

Variable p can't be reached from the main function.

Program Structure

- Declare functions before calling them for 1st time
  - Put definition before first call
  - Put declaration before first call and definition later
- Purpose: Tell the compiler the type and number of arguments to expect.
- If C does not know the function prototype before the 1st call, it automatically converts char and short to int and float to double: AVOID!

Example

```
#include <stdio.h>
double percentage (double a, double b);
//function declaration

int main (void) {
 double x = 5, y = 20;
 double val = percentage(x, y);
 return 0;
}

double percentage (double a, double b) {
 //function definition
 double p = a/b*100;
 return p;
}
```

Is this legal code?

```
int main(void) {
  char input[] = {'A', 'B', 'C', 'D'};
  char letter = get_letter(2, input);
  printf("Letter = %c", letter);
  return 0;
}
char get_letter(int x, char word[]) {
  return word[x];
}
```

- Legal
- No, array is wrong
- No, printf is wrong
- No, something else is wrong

Parameters & Arguments

- Parameters:
  - Appear in function definition
  - Represent names given to the input values
- Arguments:
  - Expressions that appear in function calls
  - Passed by value: when a function is called, arguments are copied and passed to the function

Passing by value example

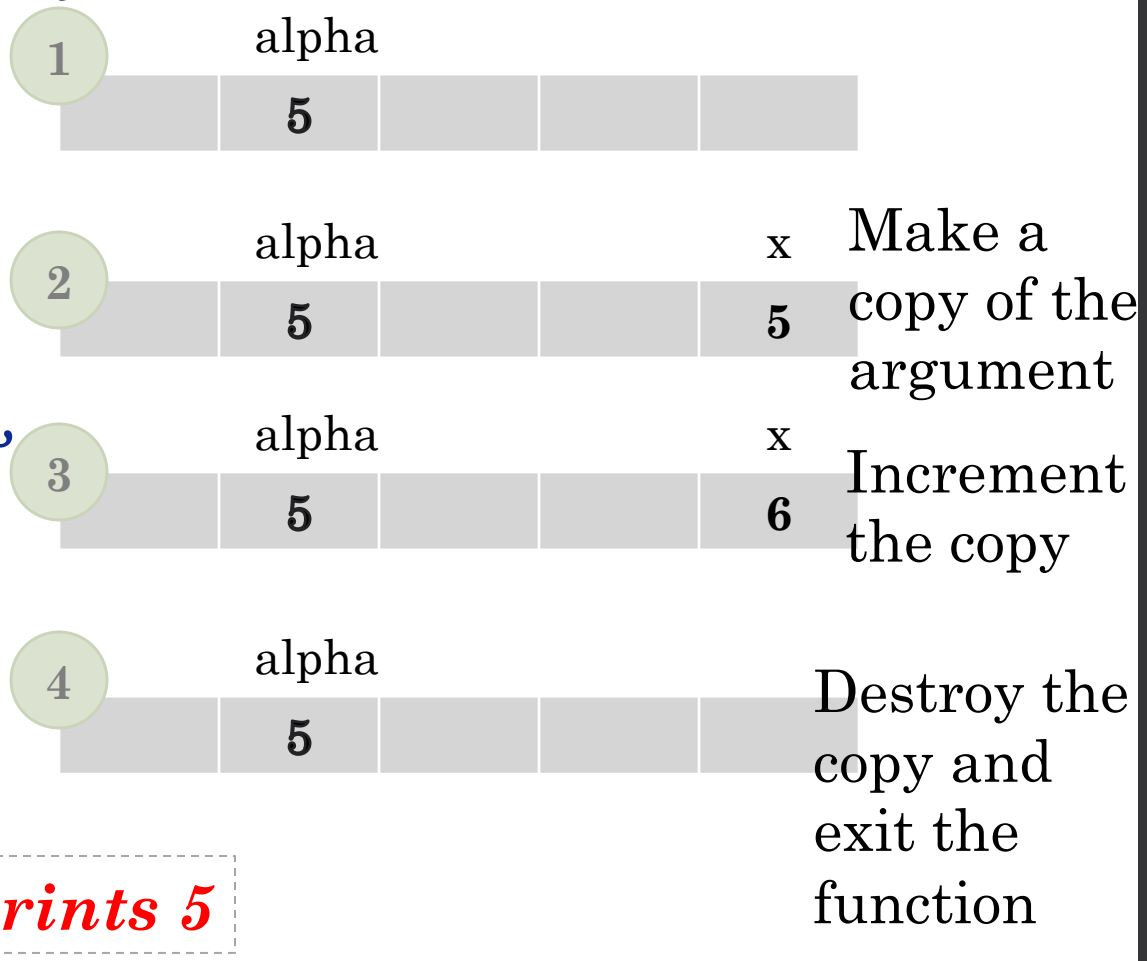
```
#include <stdio.h>

void addOne (int x) {
 x++;
}

int main (void) {
 int alpha = 5;
 addOne(alpha);
 printf("alpha is: %d", alpha);
 return 0;
}
```

Passing by value example

```
#include <stdio.h>
void addOne (int x)
  { x++; }
int main (void) {
  int alpha = 5;
  addOne(alpha);
  printf("alpha: %d",
  alpha);
  return 0;
}
```



prints 5

Advantage of passing by value

- Modify arguments inside function and still use the old value outside the function => reduce the number of variables that you need to declare inside the function

```
#include <stdio.h>
int factorial (int x) {
    int result = 1;
    while (x > 1)
        result = result*x--;
    return result;
}
```

```
int main (void) {
    int alpha = 5;
    int fact1 =
      factorial(alpha);
    int fact2 =
      factorial(alpha-1);
    printf("%d! = %d",
      alpha, fact1);
    printf("%d! = %d",
      alpha-1, fact2);
    return 0;
}
```

Array Arguments

- When one dimensional arrays are passed as arguments, leave length unspecified
- How does the function know the size of the array? Pass the size as another argument

```
void printArray(
        int a[ ], int n) {
    int x;
    for (x = 0; x < n; x++) {
                …
    }
}
```

```
int score[ ] = {1, 2,
      3, 4, 5, 6, 7};
printArray(score, 7);
printArray(score, 5);
```

No brackets in call

Returning arrays?

- How do I return an array?
- Not yet, you need to learn how pointers work first.
- Until then, you will not need to return arrays from functions.

Arrays and Pass-By-Value

- Arrays are different from scalar types, if you pass an array into a function, that function can modify the original array.
  - This is because arrays are passed as pointers (more on that later in the course)

```
#include <stdio.h>
#define LENGTH 4
void halve(int array[], int length);
int main(void) {
  int nums[LENGTH] = {2, 4, 6, 8};
  halve(nums, LENGTH);
  for (int i = 0; i < LENGTH; ++i) {
    printf("%d\n", nums[i]);
  }
}
void halve(int array[], int length) {
  for (int i = 0; i < length; ++i) {
    array[i] /= 2;
  }
}
```

More Array Arguments (Don't need to know for CSE 220)

- When multi dimensional arrays are passed as arguments, only first dimension length may be unspecified

```
void printArray(int a[ ][LEN], int n) {
    int x, y;
    for (x = 0; x < n; x++) {
      for( y=0; y<LEN; y++) {
            ...
      }
    }
}
```

What does this code output?

```
int xyz(int input[], int length) {
  int result = 0;
  for (int i = 0; i < length; ++i) {
    result += input[i];
  }
  return result;
}
int main(void) {
  int array[] = {1, 2, 3, 4};
  printf("%d", xyz(array, 4));
  return 0;
}
```

1. 0
2. 24
3. 10
4. Something Else

Recursion

- A function is recursive if it calls itself

```
int fact(int n) {
    if (n <= 1) {
        return 1;
    } else {
        return n*fact(n-1);
    }
}
```

Recursion

```
int fact(int n) {
    if (n <= 1) {
        return 1;
      } else {
        return n*fact(n-1);
      }
}
```

Termination Condition

```
x = fact(4);
x = 4 * fact(3);
x = 4 * (3 * fact(2));
x = 4 * (3 * (2 * fact(1)));
x = 4 * (3 * (2 * 1));
```

Termination Condition - Example

```
int fact(int n) {
    return n*fact(n-1);
}
```

If we remove the termination condition, the function will call itself infinite number of times.

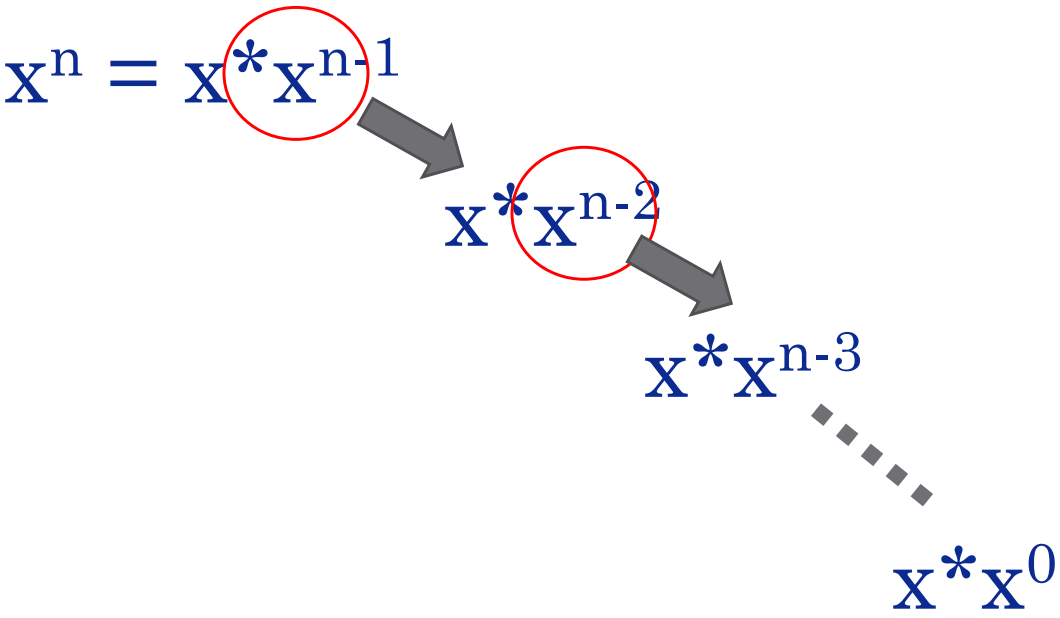
```
x = fact(4);
x = 4 * fact(3);
x = 4 * (3 * fact(2));
x = 4 * (3 * (2 * fact(1)));
x = 4 * (3 * (2 * (1 * (fact(0)))));
x = 4 * (3 * (2 * (1 * (0 * (fact(-1) ))))));
x = 4 * (3 * (2 * (1 * (0 * (-1 * fact(-2)))))));
…
```

Example

- Write a recursive function that computes x^n

$$x^n = x * x^{n-1}$$

Stopping condition?



Solution

- Write a recursive function that computes x^n

```
int power(int x, int n) {
    if (n == 0) {
        return 1;
    } else {
        return x*power(x, n-1);
    }
}
```

Exercise

- Write a recursive function prints numbers from 1 to n

```
void countToN(int start, int n) {
   printf("%d\n", start);
   if (start < n)
      countToN(start+1, n);
}
```

Exercise

- Write a recursive function that computes the sum of all integers between 1 and n

Pseudocode:
Sum(from 1 to n) = n + Sum (from to n-1)

Solution

- Write a recursive function that computes the sum of all integers between 1 and n

```
int sum (int n) {
   if (n == 1)
      return 1;
   return n + sum(n-1);
}
```

Exercise

- Write a non-recursive function that given an array of integers, counts the occurrences of the value 5 in it

```
int countFives(int array[ ], int n) {
      int count = 0;
      for (int index = 0; index < n; index++) {
            if (array[index] == 5) {
                  count++;
            }
      }
      return count;
}
```

Exercise

- Write a recursive function that given an array of integers, counts the occurrences of the value 5 in it

Number of 5s in array between 0 and n – 1 is

number of 5s in first cell
+
number of 5s between 1 and n – 1

Solution

- Write a recursive function that given an array of integers, counts the occurrences of the value 5 in it

```
int countFives(int array[ ], int start, int n) {
      int count = 0;
      if (array[start] == 5)
            count = 1;
      return count + countFives(array, start+1,
n);
}
```

Stopping Condition?

Solution

- Write a recursive function that given an array of integers, counts the occurrences of the value 5 in it

```
int countFives(int array[ ], int start, int n) {
      int count = 0;
      if (start == n) {
            return 0;
      }
      if (array[start] == 5) {
            count = 1;
      }
      return count + countFives(array, start+1, n);
}
```

Exercise

- Write a recursive function that finds the minimum number in a given array

The min in array (from 0 to n-1) = the minimum between:

- Array[n-1]
- The min in array (from 0 to n-2)

Solution

- Write a recursive function that finds the minimum number in a given array

```
int findMin(int array[ ], int n) {
   int lastIdx = n – 1;
   if (lastIdx == 0) { return array[0]; }
   int minFirstPart = findMin(array, lastIdx);
   if (array[lastIdx] < minFirstPart) {
      return array[lastIdx];
   } else {
      return minFirstPart;
   }
}
```

```
int findMin(int array[ ], int n) {
        int lastIdx = n – 1;
        if (lastIdx == 0)
                return array[0];
        int minFirstPart = findMin(array, lastIdx);
        if (array[lastIdx] < minFirstPart)
                return array[lastIdx];
        else
                return minFirstPart;
}
```

Sample Run

```
int values[3] = {2, 1, 5};
findMin(values, 3);
        lastIdx = 3 – 1 = 2
        minFirstPart = findMin(values, 2)
                        lastIdx = 2;
                        minFirstPart = findMin(values, 1)
                                        lastIdx = 0
                                        return array[0] =>  2
                        minFirstPart = 2
                        array[lastIdx] = array[1] = 1 < minFirstPart
                        return minFirstPart => 1
        minFirstPart = 1
        array[lastIdx] = array[2] = 5
        array[lastIdx] > minFirstPart        (5 > 1)
        return minFirstPart => 1
```