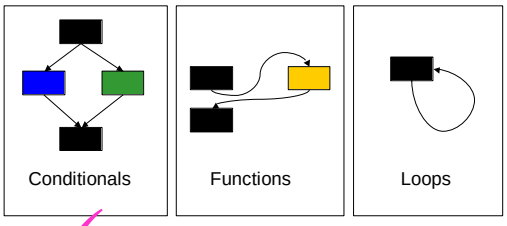
CSE 142
Programming I

Functions

Chapter 3

- Read it all!
  - ➤3.1: Reusing program parts
  - ➤3.2: Built-in math functions
  - ➤3.3: Top-down design
  - ➤3.4: Functions with no parameters
  - ➤3.5: Functions with parameters

Control Structures



What Functions Give Us

- Break the problem up into “brain-sized” chunks
- More easily reason about what the program what the program will do
- Let us write code once and use it many times
- Centralize changes

Where We’re Going to Go

1: Functions with no parameters
no return value

2: Functions with parameters
no return value

3: Functions with parameters
return value

(We’ll explain parameters and return values)

A Simple Program

```
#include <stdio.h>
int main(void)
{
    /* produce some output */
    ...
    /* print banner lines */
    printf("****************\n");
    printf("****************\n");

    /* produce more output */
    ...
    /* print banner lines */
    printf("****************\n");
    printf("****************\n");

    /* produce even more output
       */
    ...
    /* print banner lines */
    printf("****************\n");
    printf("****************\n");

    /* produce final output */
    ...
    return (0) ;
}
```

What's Wrong With This?

- What if we wanted to…
  - ➤…change the number of rows of asterisks?
  - ➤…change the number of asterisks in a row?
  - ➤…use hyphens instead of asterisks?
  - ➤…print the date with each separator?

The Big Idea Behind Functions

- Identify a sub-problem that must be solved
- Write code to solve the sub-problem *once*
- Give the code a name
- Now you can execute all of that code just by calling (or invoking) the function

What about our example?

- We keep printing banner lines over and over again
- Put those commands in a function
- Now we can change the banner style everywhere by changing the code in one place

Our example rewritten

```
#include <stdio.h>

void printBanner(void){
    printf("***************\n");
    printf("***************\n");
    return;
}


int main(void)
{
    /* produce some output */
    ...
    /* print banner lines */
    printBanner();

    /* produce more output */
    ...
    /* print banner lines */
    printBanner();

    /* produce even more output
       */
    ...
    /* print banner lines */
    printBanner();

    /* produce final output */
    ...
    return (0) ;
}
```


Anatomy of a Simple Function

```
void functionName(void){

  /* local variable decls. */

  /* code */

  return;
}
```


Our Function

```
void printBanner(void){
  /* local variable decls. */

  /* code */
  printf("***************\n");
  printf("***************\n");

  return;
}
```

Local Variables

- Each function can have variables declared inside of it, these are local variables
- These variables *do not* have any meaning outside of the function
- Several functions can have variables with the same name *they are different variables*
  - ➤Each function has it's own name space

A Silly Example

```
void silly(void){
  int foo;
  scanf("%d", &foo);
  printf("You entered %d!\n", foo);
  return;
}
```


Shortcomings So Far

- No communication from caller to the function
  - ➤The function cannot see variables in main!
  - ➤Fix this with parameters
- No communication from the function to the caller
  - ➤Fix this with the return value

Example With Parameters

```
void printMonth(int month){
   switch (month){
      case 1:   printf("January");
                break;
      case 2:   printf("February");
                break;
      /* etc. */
   }
   return;
}
```


New Syntax

`void functionName(`*parameter list*`){...}`

- The parameter list is a comma separated list of variables names *with their type*s:

```
(int foo, double bar)

(int xVal, int yVal)
```


How Do We Call This Function?

- Now that we have the function printMonth() we want to use it.
- What will these do?

```
printMonth(1);

printMonth(2*2);

printMonth();
```

Parameters and Arguments

- Parameters act like local variables except that they are initialized with the values that the function was called with
- The values passed in are called arguments

Another Example

```
#include <stdio.h>

void printArea(double
  length, double width){

  double area;

  area = length * width;
  printf("Area = %f\n",
      area);
  return;
}
```

```
int main(void){
  double x, y;

  printf("Length: ");
  scanf("%lf", &x);
  printf("Width: ");
  scanf("%lf", &y);

  printArea(x, y);

  return 0;
}
```


Cautions!

- The parameters and arguments should match in number, type, and order!
- The book uses different terminology
  parameter = formal parameter
  argument = actual parameter
- Functions must be declared before they are used!
  - ➤(We'll see how to get around this in just a bit.)

Local Variables and Parameters

- Each time you call a function you get new copies of each local variable—old values *cannot* be carried over!
- Inside a function, local variables and parameters behave exactly the same except:
  - ➤Local variables start uninitialized
  - ➤Parameters are initialized with their arguments

Return Values

- A function can return a value to its caller

```
double circleArea(double radius){…}
```

return type — function name — argument list — function body

return

- The `return` command terminates and sets the value of the function
- If the return type is void, then return takes no expression
- If the return type is not void, then return *must* be given an expression

Example

```
…
int square(int number){
  return number * number;
}
…
y = square(2) * PI;
```


More on return

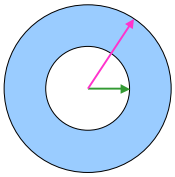
- A function can only return one value
- A function can have multiple return statements
- Which one gets used?

Example

- Write a function that finds the absolute value of a double:

One Function Can Call Another

- Let's write a function to calculate the area of a washer:



Strategy:

- This problem has a sub-problem
  - ➤Calculate the area of a disk
- The total algorithm:
  - ➤Calculate the area of the outer disk
  - ➤Calculate the area of the inner disk
  - ➤Subtract

Solve the Sub-Problem

- Find the area of a disk:

Solve the Full Problem

- Find the area of the washer:

```
double washerArea(double inside,
  double outside){




}
```


How This Interacts With main()

- Now we can find the area of a washer

```
int main(void){
  double area;
  area = washerArea(1.0, 3.0);
  printf("Area: %f\n", area);
  return 0;
}
```


Abstraction

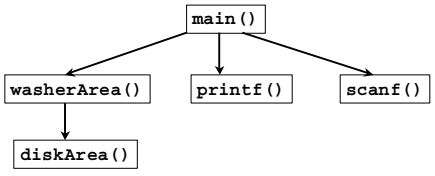
- In this example, main() called washerArea() which called diskArea() to compute the value desired
- What does main() need to know about *how* that calculation was carried out?
- This is called abstraction

main() is a Special Function

- Main is a function just like any other
- Main is special because it is automatically run when your program starts
- What is the difference between return and exit()?
- Why does main() return a value?

Static Call Graph

- Shows how functions are related



What Exactly Is Void?

- If a function return void, then it passes no value back to its caller
- If a function's parameter list is void, then it takes no arguments

Function Prototypes

- So far, if we want to use a function we must first provide a complete definition of it
- This is a pain
- We can instead give a prototype of the function, and define the function later

Prototype Example

```
double diskArea(double radius);

/* we can now use diskArea() */
double washerArea(double out, double in){
  return diskArea(out) - diskArea(in);
}

double diskArea(double radius){
  return radius * radius * PI;
}
```


Header Files

- A header file is a file that contains function prototypes, constant declarations, etc.
- stdio.h is a header file

Built-In Functions

- C has some built-in functions
  - Standard library
- To use them you must #include the correct header files
  - I/O (printf, scanf, etc.) — stdio.h
  - Utility functions (exit, etc.) — stdlib.h
  - Math functions (sin, sqrt, etc.) — math.h
  - etc., etc., etc.

printf() and scanf() Revisited

- Both printf() and scanf() have return values
- printf() returns an integer: the number of characters output
- scanf() returns an integer: the number of variables assigned
- How might you use these?

Global Variables

- C lets you define global variables: variables that are not in any function.
- In this course, they are not allowed!
  - Only local variables are allowed
  - #define constants are not variables
- Global variables have their uses, but more often they are a crutch and contribute to bad style

Functions Summary

- May take several parameters, or none.
- May return one value, or none.
- Functions help structure a program
  - ➤Perform repeated action easily
  - ➤Help you more easily reason about your program
  - ➤Abstract a service