Computer Science 5C
Chapter 4--Functions and Structured Programming

Dr. Scott A. Brandt
Professor, Computer Science Department
Director, UCSC/LANL Institute for Scalable Scientific Data Management
A program is made up of one or more *functions*.
- main() is a function
- Programs can have other functions!

To run a function, you call (invoke) the function:
- The compiler recognizes a function definition or call when it sees an identifier followed by parentheses.
- It knows which it is based on where they occur.
Function invocation II

• When a function is called
  1. The calling function pauses
  2. The function runs
  3. Control is returned to the calling function

```c
int main(void)
{
    print_message();
}

void print_message(void)
{
    printf("Hello, world\n");
}
```
Function definitions

• Functions must be defined before they can be used
• Some functions are predefined for us
  • printf()
  • scanf()
• We can define our own functions

  type function_name ( parameter list)
  {
    declarations
    statements
  }
Functions with parameters

```c
void print_message(int); /* function prototype */

int main(void)
{
    int how_many;

    printf(“How many times?\n”);
    scanf(“%d”, &how_many);

    print_message(how_many); /* function call (invocation) */
}

void print_message(int n) /* function definition */
{
    int i;

    printf(“Here is the message\n”);
    for(i = 0; i < n; i++)
        printf(“%d - Have a nice day\n”, i);
}
```
The return statement

• The return statement ends the function and sends control back to the calling function

• It can be called two ways
  • return;
  • return expression;

• A function can have zero or more return statements
  • The first one encountered causes the function to stop executing and control to return
Example

```c
int min(int a, int b);

int main(void)
{
    int j = 7, k = 5, minimum;

    minimum = min(j, k);

    printf("The minimum of %d and %d is %d\n", j, k, minimum);
}

int min(int a, int b)
{
    if(a < b)
        return a;
    else
        return b;
}
```
Function prototypes

• Function prototypes tell the compiler about
  • The function name
  • The function type
  • The number and type of parameters the function expects

• Function prototypes look just like function definitions, except
  • They have a ; in place of the {declarations and statements}
  • The parameters don’t need to have names
Top-down design

• It is useful to decompose the program into steps
• Each step may be a function
• Those steps may themselves be broken down into smaller steps
• Until we get to steps simple enough to program directly
Running sum program

• Print Banner
• Print the headings over the columns
• Read the data
• Print it neatly in columns
Debugging a program

• Two useful tools:
  • printf()
  • assert()

• printf() is used to print out values of variables
  • printf(“i = %d\n”, i);

• assert() is used to state something you believe to be true
  • assert(i > 0);
Random numbers

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

int max(int a, int b);
int min(int a, int b);
void print_random_numbers(int k);

int main(void)
{
    int n;
    printf(“How many numbers shall I print?”);
    scanf(“%d”, &n);
    print_random_numbers(n);
}
void print_random_numbers(int k)
{
    int i, r, biggest, smallest;
    r = biggest = smallest = rand();
    printf("\n%7d", r);
    for(i = 1; i < k; i++) {
        if(i % 7 == 0) {
            printf("\n");
            r = rand();
            biggest = max(r, biggest);
            smallest = min(r, smallest);
            printf("%7d", r);
        }
    }
    printf("\n\n%s%5d
%s%5d
%s%5d
\n", "   Count: ", k,
            "Maximum: ", biggest,
            "Minimum: ", smallest);
}