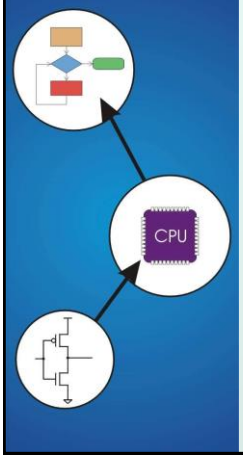


# Chapter 14
# Functions

Original slides from Gregory Byrd, North Carolina State University

Modified slides by Chris Wilcox, Colorado State University



## Functions

- **Smaller, simpler, subcomponent of program**
- **Provides abstraction**
  - hide low-level details, give high-level structure
  - easier to understand overall program flow
  - enables separable, independent development
- **C functions**
  - *not* methods—no objects, here!
  - zero or multiple arguments passed in
  - single result returned (optional)
  - return value is always a particular type
- In other languages, called procedures, routines, ...





## Example of High-Level Structure

```
int main()
  {
    SetupBoard(); /* place pieces on board */
    DetermineSides(); /* choose black/white */

    /* Play game */
    do {
      WhitesTurn();
      BlacksTurn();
    } while (NoOutcomeYet());
  }
```

Structure of program is evident, even without knowing implementation.





## Functions in C

- **Declaration** (also called prototype)

```
int Factorial(int n);
```

type of return value | name of function | types of all arguments

- **Function call** -- used in expression

```
a = x + Factorial(f + g);
```

1. evaluate arguments
2. execute function
3. use return value in expression

## Function Definition

- State type, name, types of arguments
  - must match function declaration
  - give name to each argument (doesn't have to match declaration)

```
int Factorial(int n)
{
  int i;
  int result = 1;
  for (i = 1; i <= n; i++)
     result *= i;
  return result;
}
```

gives control back to calling function and returns value





## Why Declaration?

- Since function definition also includes return and argument types, why is declaration needed?
- **Use might be seen before definition.**
  Compiler needs to know return and arg types and number of arguments.
- **Definition might be in a different file, written by a different programmer.**
  - include a "header" file with function declarations only
  - compile separately, link together to make executable





## Example

```
double ValueInDollars(double amount, double rate);
int main()
{
    ...
    dollars = ValueInDollars(francs,
                             DOLLARS_PER_FRANC);
    printf("%f francs equals %f dollars.\n",
           francs, dollars);
    ...
}
double ValueInDollars(double amount, double rate)
{
  return amount * rate;
}
```

function declaration (prototype)

function call (invocation)

function definition (code)





## Implementing Functions: Overview

- Activation record (stack frame)
  - information about each function, including arguments and local variables
  - stored on run-time stack