CSE101-lec 11

Designing Structured Programs

Introduction to Functions

Divide and Conquer

- Best way to solve a problem is by dividing the problem and solving it.
- **Divide and conquer**
  - Construct a program from smaller pieces or components
    - These smaller pieces are called **modules**
  - Each module more manageable than the original program

Program Modules in C

- Functions
  - Modules in C are called functions.
  - Programs combine user-defined functions with library functions
    - C standard library has a wide variety of functions for performing common *mathematical calculations*, *string manipulations*, *character manipulations*, *input/output* and many more.
    - C standard library makes your job easier.
    - Functions like printf(), scanf(), pow() are standard library functions.
    - We can also write functions to define some specific task in a program and these functions are called user-defined functions.

Functions

- Functions
  - Modularize a program
  - All variables defined inside functions are local variables
    - Known only in function defined.
  - Parameters
    - Functions have list of parameters.
    - Communicate information between functions.
    - Are also Local variables to that function.

Benefits of functions

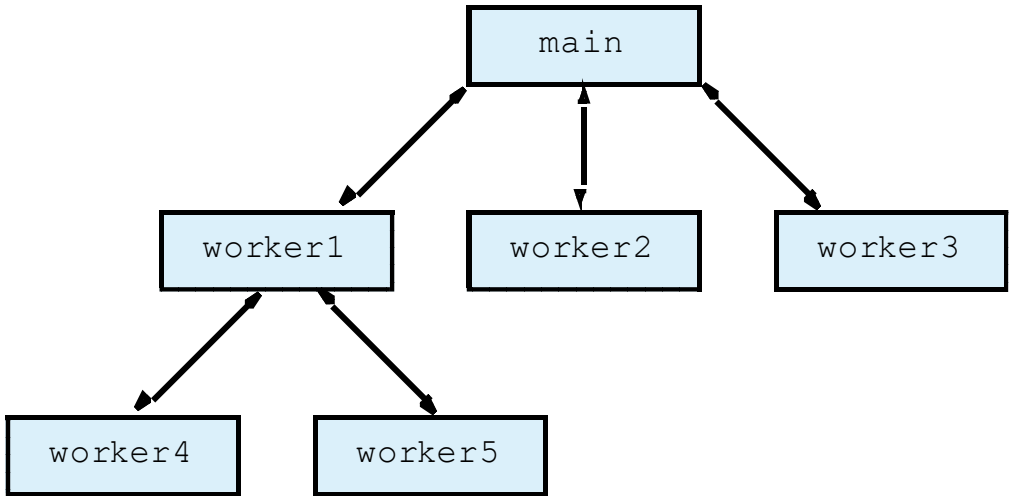
- Divide and conquer
  - Manageable program development
- Software reusability
  - Use existing functions as building blocks for new programs
  - Abstraction - hide internal details (library functions)
- Avoid code repetition

Function Call

- Function calls
  - Invoking functions
    - Provide function name and arguments (data)
    - Function performs operations or manipulations
    - Function returns results
  - Function call analogy:
    - Boss asks worker to complete task
      - Worker gets information, does task, returns result
      - Information hiding: boss does not know details

Program Modules in C

Hierarchical boss function/worker function relationship.



Function Definitions

- Function definition format

  return-value-type function-name(parameter-list)
  {
  declarations and statements
  }

  - Function-name: any valid identifier
  - Return-value-type: data type of the result (default `int`)
    - `void` – indicates that the function returns nothing
  - Parameter-list: comma separated list, declares parameters
    - A type must be listed explicitly for each parameter unless, the parameter is of type `int`

Function Definitions

- Function definition format (continued)

  return-value-type function-name(parameter-list)
  {
  declarations and statements
  }

  - Definitions and statements: function body (block)
    - Variables can be defined inside blocks (can be nested)
    - Functions can not be defined inside other functions
  - Returning control
    - If nothing returned
      - `return;`
      - or, until reaches right brace at the end of function.
    - If something returned
      - `return` *expression ;*

Function Prototypes

- Function prototype
  - Function name
  - Parameters – what the function takes in
  - Return type – data type function returns (default `int`)
  - Used to validate functions
  - Prototype only needed if function definition comes after use in program
  - The function with the prototype

    ```
    int square( int y);
    ```

    - Takes in 1 int data.
    - Returns an `int`
- Promotion rules and conversions
  - Converting to lower types can lead to errors

```c
#include <stdio.h>
int square(int y); // function prototype
int main()
{
    int x; //counter
    for ( x = 1; x <= 10; ++x) {
       printf( "%d  ", square(x));  //function call
    } //end for
   puts(" ");
} //end main

int square( int y ) // function definition
{
    return y * y;  //returns the square of y as an int
}
```

```
1   4   9   16   25   36   49   64   81   100
```