

## Chapter 11 Introduction to Programming in C

Original slides from Gregory Byrd, North Carolina State University  
Modified slides by Chris Wilcox, Colorado State University

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## C: A High-Level Language

- ◆ **Gives symbolic names to values**
  - don't need to know register or memory location
- ◆ **Provides abstraction of underlying hardware**
  - operations do not depend on instruction set
  - example: "a = b \* c", even without multiply instruction
- ◆ **Provides expressiveness**
  - use meaningful symbols that convey meaning
  - simple expressions for control patterns (if-then-else)
- ◆ **Enhances code readability**
- ◆ **Safeguards against bugs**
  - enforce rules or conditions at compile-time or run-time

CS270 - Spring Semester 2014 2

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## Compilation vs. Interpretation

- ◆ Different ways of translating high-level language
- ◆ **Interpretation**
  - interpreter = program that executes program statements
  - generally one line or command at a time
  - limited scope of processing
  - easy to debug, make changes, view intermediate results
  - languages: BASIC, LISP, Perl, Java, Matlab, C-shell
- ◆ **Compilation**
  - Compiler = program that makes an executable from code
  - translates statements into machine language
  - performs optimization over multiple statements
  - change requires recompilation
  - optimized code can be harder to debug
  - languages: C, C++, Fortran, Pascal

CS270 - Spring Semester 2014 3

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

## Compilation vs. Interpretation

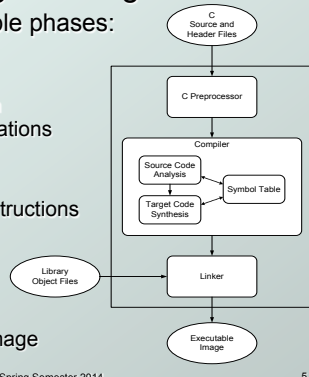
- ◆ Consider the following algorithm:
  - Get **W** from the keyboard.
  - $X = W + W$
  - $Y = X + X$
  - $Z = Y + Y$
  - Print **Z** to screen.
- ◆ If interpreting, how many arithmetic operations?
- ◆ If compiling, can we simplify the computation?
- ◆ Yes, by analyzing the entire program, we can reduce to single arithmetic operation!

CS270 - Spring Semester 2014 4

## Compiling a C Program

Compilers have multiple phases:

- **Preprocessor**
  - macro substitution
  - conditional compilation
  - source-level transformations
  - output is still C code
- **Compiler**
  - generates machine instructions
  - output is object file
- **Linker**
  - combines object files (including libraries)
  - output is executable image



## Compiler

- **Source Code Analysis**
  - “front end”
  - parses programs to identify its pieces: (variables, expressions, statements, functions, etc.)
  - depends on language, not on target machine
- **Code Generation**
  - “back end”
  - generates machine code from analyzed source
  - may optimize machine code for efficiency
  - very dependent on target machine
- **Symbol Table**
  - map between symbolic names and items
  - like assembler, but more kinds of information

## A Simple Java Program

```

import java.io.*;
public class Simple {
    /* Function: main */
    /* Description: counts down from user input to STOP */
    public static void main(String[] args) ...
    {
        /* variable declarations */
        public static final int STOP = 0;
        int counter; /* an integer to hold count values */
        int startPoint; /* starting point for countdown */

        /* prompt user for input, assumes scanner */
        System.out.println("Enter a positive number: ");
        startPoint = in.nextInt();

        /* count down and print count */
        for (counter=startPoint; counter>=STOP; counter--)
            System.out.println(counter);
    }
    
```

## A Simple C Program

```

#include <stdio.h>
#define STOP 0
/* Function: main */
/* Description: counts down from user input to STOP */
int main(int argc, char *argv[])
{
    /* variable declarations */
    int counter; /* an integer to hold count values */
    int startPoint; /* starting point for countdown */

    /* prompt user for input */
    printf("Enter a positive number: ");
    scanf("%d", &startPoint); /* read into startPoint */

    /* count down and print count */
    for (counter=startPoint; counter>=STOP; counter--)
        printf("%d\n", counter);
}
    
```

## Preprocessor Directives

- ◆ **#include** <stdio.h>
  - Before compiling, copy contents of **header file** (stdio.h) into source code.
  - Header files typically contain descriptions of functions and variables needed by the program.
  - No restrictions, could be any C source code, including your own.
- ◆ **#define** STOP 0
  - Commonly called a **macro**, before compiling, replace all instances of string "STOP" with "0"
  - Used for values that are constant during execution, but might change if the program is reused. (requires recompilation.)

## Comments

- ◆ Begins with **/\***, ends with **\*/**
- ◆ Can span multiple lines
- ◆ Cannot have a comment within a comment
- ◆ -c99 allows use of single line comments: **//**
- ◆ Comments are not recognized within a string
  - example: "my/\*don't print this\*/string"  
would be printed as: my/\*don't print this\*/string
- ◆ **As before, use comments to help reader, not to confuse or to restate the obvious**

## main Function

Every C program must have a **main()** function:

- ◆ The main function contains the code that is executed when the program is run.
- ◆ As with all functions, the code for main lives within brackets:

```
main(int argc, char *argv[])  
{  
    /* code goes here */  
}
```

- ◆ Very similar to Java, but need a size of array since C does not have length member.

## Variable Declarations

- ◆ Variables are used as names for data items.
- ◆ Each variable has a **type**, which tells the compiler how the data is to be interpreted (and how much space it needs).  
**int counter;**  
**int startPoint;**
- ◆ **int** is a predefined signed integer type in C.

## Input and Output

Variety of I/O functions in *C Standard Library*:

- Must include `<stdio.h>` to use them.
  - `printf("%d\n", counter);`
    - String contains characters to print and formatting directions for variables.
    - This call prints the variable `counter` as a decimal integer, followed by a newline (`\n`).
  - `scanf("%d", &startPoint);`
    - String contains formatting directions for interpreting the type of the input.
    - This call reads a decimal integer and assigns it to the variable `startPoint`. (Don't worry about the `&` yet!)

## More About Output

- Can print arbitrary expressions, not just variables
  - `printf("%d\n", startPoint - counter);`
- Print multiple expressions with a single statement
  - `printf("%d %d\n", counter, startPoint - counter);`
- Different formatting options:
  - `%d` decimal integer
  - `%x` hexadecimal integer
  - `%c` ASCII character
  - `%f` floating-point number

## Examples

- This code:

```
printf("%d is a prime number.\n", 43);
printf("43 plus 59 (decimal) is %d.\n", 43+59);
printf("43 plus 59 (hex) is %x.\n", 43+59);
printf("43 plus 59 (char) is %c.\n", 43+59);
```

produces this output:

```
43 is a prime number.
43 + 59 in decimal is 102.
43 + 59 in hex is 66.
43 + 59 as a character is f.
```

## Examples of Input

- Many of the same formatting characters are available for user input.
  - `scanf("%c", &nextChar);`
    - reads a single character and stores it in `nextChar`
  - `scanf("%f", &radius);`
    - reads a floating point number and stores it in `radius`
  - `scanf("%d %d", &length, &width);`
    - reads two decimal integers (separated by whitespace), stores the first one in `length` and the second in `width`
- Must use ampersand (`&`) for variables being modified, pointers will be discussed later.

## Compiling and Linking

- ◆ Various compilers available
  - cc, gcc
  - includes preprocessor, compiler, and linker
- ◆ Lots and lots of options!
  - level of optimization, debugging
  - preprocessor, linker options
  - usually controlled by makefile
  - intermediate files --  
object (.o), assembler (.s), preprocessor (.i), etc.

## Remaining Chapters

- ◆ A more detailed look at many C features:
  - Variables and declarations
  - Operators
  - Control Structures
  - Functions
  - Data Structures
  - I/O
- ◆ Emphasis on how C is converted to assembly language.
- ◆ Also see C Reference in Appendix D.

## C Example

```
#include <stdio.h>
#define STOP 0
/* Function: main */
/* Description: counts down from user input to STOP */
main()
{
    /* variable declarations */
    int counter; /* an integer to hold count values */
    int startPoint; /* starting point for countdown */
    /* prompt user for input */
    printf("Enter a positive number: ");
    scanf("%d", &startPoint); /* read into startPoint */
    /* count down and print count */
    for (counter=startPoint; counter>=STOP; counter--)
        printf("%d\n", counter);
}
```