Principles of Computer Science II

Nadeem Abdul Hamid
CSC121A - Spring 2005

Lecture Slides 8 - Computer Science & the C Programming Language

Elements of Computer Science

- Mathematics
  - Use formal languages to denote ideas
- Engineering
  - Design
  - Assemble components into systems
  - Evaluate tradeoffs among alternates
- Natural Science
  - Observe behavior of complex systems
  - Form hypotheses
  - Test predictions

Portions of this lecture from *How to Think Like a Computer Scientist* by Allen B. Downey

Problem-Solving

- Single most important skill
  - Ability to formulate problems
  - Think creatively about solutions
  - Express a solution clearly and accurately
- Learning to program = Developing problem-solving skills

C Programming Language

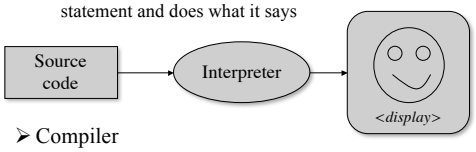
- C: Intermediate-level language
  - Java: high-level
  - Assembly/machine language: low-level
  - Loosely, computer only execute low-level code
  - High-level programs must be translated to low-level before they are run

High-Level Languages

- Disadvantage
  - Must be translated before can be run (takes time)
- Advantages
  - Much easier to program
    - Less time to write
    - Shorter/easier to read
    - More likely to be correct
  - Portable
    - Can run on different kinds of computers with little/no modification

"Translating" a Program

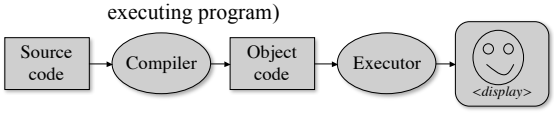
- Interpreter
  - Program that reads a high-level program statement by statement and does what it says



- Compiler
  - Program that reads a high-level program and translates to machine code all at once before running it

Compiler

- Using a compiler
  - Use text editor to write program, hello.c (save to disk)
  - Compile program to object, hello.o or executable, hello.exe (save to disk)
  - Run program (executor loads program from disk into memory and makes computer start executing program)



What is a Program?

- A sequence of instructions specifying how to perform a computation
- A concrete implementation of an algorithm for a computer
- Basic functions of any language
  - Input
  - Output
  - Math
  - Testing (Selection/Branching)
  - Repetition (Looping/Iteration)
  - (Subroutines)
- Programming: Process of breaking a large, complex task into smaller and smaller subtasks until they can be performed by basic functions

Debugging

To err is human;
To really screw things up takes a computer.

- Bugs: program errors
- Tracking them down and correcting them: debugging
- Types of errors
  - Compile-time (mostly syntactic)
  - Run-time
  - Logic/Semantic

Syntactic Errors

- Compiler can only translate a syntactically-correct program
- Syntax: structure of the program and rules about that structure
  - English syntax: Sentence must begin with a capital letter and end with period
    - this sentence contains a syntax error.
    - So does this one
- We can process syntax errors without spewing error messages; compilers aren't like that
- More syntax rules in C than English?

Logic/Semantic Errors

- Program compiles, may even run to completion without generating error messages
- Program does not do what you *mean* it to do
- Program you wrote is not the program you wanted to write
  - The meaning of the program (its semantics) are wrong

Developing and Debugging

- Though frustrating, debugging is one of most intellectual, challenging, interesting parts of programming
- Like detective work…
- Like experimental science…
  - Figure out what's wrong, make a change, try it out
- Developing a program should be an incremental process
  - Start with working program that does *something*
  - Make small modifications, compiling & debugging as you go… always have a working program
  - Linux started as a simple program of Linus Torvalds to explore the Intel 80386 chip
    - An early project: A program to switch between printing AAAA and BBBB… later evolved to Linux OS

Languages

- Natural: languages people speak… evolved naturally, not designed persay
- Formal: languages designed by people for specific applications
  - Mathematical notation
  - Chemical notation
  - Programming languages: formal languages designed to express computations
- Figuring out tokens and structure: *parsing* (we do this unconsciously for English)
- After parsing, we figure out the meaning (semantics)

13

Natural and Formal Languages

- Similarities: Basic concepts of tokens, structure, syntax, and semantics
- Differences
  - Ambiguity
  - Redundancy
  - Literalness
- Formal languages much more dense and concise
  - Structure is very important… reading top to bottom, left to right doesn't always work
  - Details matter! More picky than an English teacher about spelling errors and punctuation

14

First C Program

```
/* The traditional first program in honor of
   Dennis Ritchie, who invented C while
   at Bell Labs in 1972. */

#include <stdio.h>

int main(void)
{
   printf("Hello, world!\n");
   return 0;
}
```

Comments in C enclosed in /* … */
(// form is not in ANSI C)

#… : preprocessing directive
This one causes a copy of the standard header file for input/output to be included at this point in the code
<…> (angle brackets) indicate file is to be found in the "usual place"
In this case, we need the information about the `printf` function

Every program has a function named `main`. Parentheses after name indicate to the compiler that it is a function. Keyword `int` declares return type. Keyword `void` indicates that it takes no arguments

Here the `printf` (print-formatted) function is being *called*, or *invoked*, with a single argument - a string constant (C functions are similar to Java methods)

15

C Compiler for Windows/DOS

- http://www.delorie.com/djgpp/
  - Zip Picker
    - Basic functionality: "Build and run programs with DJGPP"
    - Programming language: "C"
    - Integrated Development Env. and Tools: "None"
  - Downloads a bunch of ZIP files - uncompress to C:\DJGPP directory
  - Set up path and other environment variables as:
    - http://www.delorie.com/djgpp/doc/ug/intro/installing-djgpp.html
- Can use SciTE text editor to edit
- Use command window (terminal) to compile and run programs

16

Variables, Expressions, Assignments

```
#include <stdio.h>

int main(void)
{
   int   inches, feet, fathoms;

   fathoms = 7;
   feet = 6 * fathoms;
   inches = 12 * feet;
   printf("Wreck of the Hesperus:\n");
   printf("Its depth at sea in different units:\n");
   printf("   %d fathoms\n", fathoms);
   printf("   %d feet\n", feet);
   printf("   %d inches\n", inches);
   return 0;
}
```

Variable declarations

Assignment statements

First argument to `printf` is always a string, called the *control string*.

`%d` is a 'conversion specification' or *format*. In this case, it specifies that the second argument, `fathoms`, should be printed in the format of a decimal integer.

17

Simple Layout of C Program

preprocessing directives

int main(void)
{
  declarations

  statements
}

What kind of data can be stored in each variable. Compiler sets aside appropriate amount of memory for it.

Carry out input, output, and desired computations

18

Program Elements

- Variables: referred to by an *identifier* (i.e. a name)
  - Identifier consists of letters, digits, underscore; cannot begin with digit
  - Should be named to reflect use in the program
  - After being declared, can be assigned values and used in expressions
- Keywords, also called *reserved words*
  - Cannot be used as the names of variables
  - Will be presented in Chapter 2

19

Program Elements

- Expressions
  - Typically found (1) to the right of assignment operators or (2) as arguments to functions
  - Constants… 6 12
  - Name of a variable alone
  - Meaningful combinations of operators with variables and constants (or other expressions)
  - Basic arithmetic operators: `+ - * / %`
- Assignment statements
  - Variable on left side, equal sign =, expression on right
  - Expression can be simple or complex and contain function calls
  - Constants and most expressions not allowed on left of =

20

Characters

- Variables and constants of type `char` are used to manipulate characters
- Constants written within single quotes
- Printf format: `%c`

```
#include <stdio.h>

int main(void)
{
   char  c;
   c = 'A';
   printf("%c\n", c);     /* the letter A is printed */
   return 0;
}
```

21

Floating Point Types

- Three types in ANSI C:
  - `float`, `double`, `long double`
- *Working floating type* in C is `double`
  - `1.065` `0.004` `7.0`
- Float constants specified with `F` suffix
  - `1.065F` `0.004F`
- `Long double` specified with `L` suffix
- Printf format: `%f` (`float` number with 6 digits to the right of decimal point)
- Modulus operator % works only with integers

22

Initialization

- Variables may be initialized when declared

```
int   fathoms = 7, feet = 6 * fathoms, inches = 12 * feet;
```

- Variables cannot be used before they are declared

23

Preprocessing Directives

- When C compiler is invoked, the first thing that executes is the preprocessor
- Preprocessor modifies the source code that is passed to the compiler
  - Contents of other files may be copied in
  - Specified character strings replaced with others
- Lines beginning with # give commands to the preprocessor and are called *preprocessing directives*
  - Can occur anywhere in a program
  - Affects only those lines coming after the directive

24

#define and #include

```
#include <filename>
```

- Preprocessor looks for *filename* in standard directories (in Unix, typically /usr/include)

```
#include "filename"
```

- Preprocessor looks in current directory and then other directories (usually in the path)

```
#define LIMIT 100
#define PI    3.14159
```

- Preprocessor changes all occurrences of `LIMIT` to `100` and `PI` to `3.14159` (other than things in string constants)

```
printf("PI = %f\n", PI);   ➠   printf("PI = %f\n", 3.14159);
```

- `#define`'d things are called *symbolic constants*

Using printf

```
printf("Get set: = %s %d %f %c%c\n",
       "one", 2, 3.33, 'G', 'O' );
```

- … prints out:

```
Get set: one 2 3.330000 GO
```

- Conversion characters:
  - c - character
  - d - decimal integer
  - e - f.p. in scientific notation
  - f - f.p. number
  - g - the shorter of the e or f format
  - s - string

Field Width and Precision

- *Field*: The place where an argument is printed
- To control *field width*, specify integer between the % and conversion character

```
printf("%c%3c%7c\n", 'A', 'B', 'C');
```

- For floating point, the *precision* (# of digits after the point) specified by *n* in `%m.nf`

```
printf("%s%.1f %.2f %.3f\n%s%7.1f%7.2f%7.3f\n",
       "Some numbers: ", 1.0, 2.0, 3.0,
       "More numbers:", 4.0, 5.0, 6.0);
```


Input with scanf

- Analogous to printf
  - First argument is control string specifying expected input types
  - Following arguments are *addresses* of variables in which to store input values
  - Address: place in memory at which a variable's data is stored
  - Symbol & represents the *address operator*

```
int x;
...
scanf("%d", &x);
```


scanf Conversion Characters

- c - character
- d - decimal integer
- f - floating-point number (`float`)
- lf - floating-point number (`double`)
- Lf - floating-point number (`long double`)
- s - string

Sample Program with Input

```
#include <stdio.h>

int main(void)
{
   char   first, middle, last;
   int    age;

   printf("Input your three initials and your age:  ");
   scanf("%c%c%c%d", &first, &middle, &last, &age);
   printf("\nGreetings %c.%c.%c.  %s %d.\n", first,
      middle, last, "Next year your age will be", age + 1);
   return 0;
}
```

Note: This program does not skip whitespace upon input

Area of a Circle

Symbolic constant

Notice %lf and %f formats

```
#include <stdio.h>

#define   PI   3.141592653589793

int main(void)
{
   double   radius;

   printf("\n%s\n\n%s",
      "This program computes the area of a circle.",
      "Input the radius:  ");
   scanf("%lf", &radius);
   printf("\n%s\n%s%.2f%s%.2f%s%.2f\n%s%.5f\n\n",
      "Area = PI * radius * radius",
      "     = ", PI, " * ", radius, " * ", radius,
      "     = ", PI * radius * radius);
   return 0;
}
```


Return Values `printf`, `scanf`

- `printf`: number of characters printed
- scanf: number of successful conversions
  - 0 if none - usually due to invalid input character (alphabetic character when number expected)
  - -1 (EOF) - if input failure, such as end-of-file, occurs before any conversion

`while` Statement

Testing the return value of scanf - we expect exactly one successful input conversion to take place

To end loop, must type a non-number or else type an "end-of-file" signal.

UNIX eof: <return> followed by <ctrl>+<d> key
Windows/DOS eof: <ctrl>+<z>

```
/* Sums are computed. */

#include <stdio.h>

int main(void)
{
   int     cnt = 0;
   float   sum = 0.0, x;

   printf("The sum of your numbers"
          " will be computed\n\n");
   printf("Input some numbers:  ");
   while (scanf("%f", &x) == 1) {
      cnt = cnt + 1;
      sum = sum + x;
   }
   printf("\n%s%5d\n%s%12f\n\n",
      "Count:", cnt,
      "  Sum:", sum);
   return 0;
}
```

(Try inputs: 1.1 2.02 3.003 4.0004 5.00005)

Programming Style

- Good coding style essential for facilitating
  - Readability,
  - Writability, and
  - Maintenance of program code.
- Use white space and comments to make code easier to read and understand, and visually attractive
- Use consistent indentation (you know that already…)

Common Programming Errors

- Easy to become confused about error messages
- Simple syntax… ; , " ' (), misspelling, etc.
- Be sure to use `%lf` with `scanf` when reading in a `double` value (with `printf` you can use `%f`)
- In `printf` format of *`%m.nf`*, the *m* specifies total field width, not the number of decimal digits to the left
  - For two digits to the left and three to the right:use `%6.3f` and *not* `%2.3`
- !!! Don't forget the address operator `&` when using `scanf` !!!

System Issues

- Interrupting a program: use <ctrl> + <c> or <ctrl> + <break>
- Typing 'End-of-File' signal
  - Unix: type return and then <ctrl> + <d>
  - Windows/DOS: <ctrl> + <z>

Redirection of Input/Output

- Many OSs allow you to *redirect* the standard input (usually connected to keyboard) and standard output (usually connected to the screen)
- By default, `printf` writes to standard output
- `scanf` reads from standard input
- Use > symbol to redirect output
- Use < symbol to redirect input

37

Double Echo Program

```
#include <stdio.h>

int main(void)
{
   char   c;

   while (scanf("%c", &c) == 1) {
      printf("%c", c);
      printf("%c", c);
   }
   return 0;
}

/* Four ways to run this program:

   dbl_out
   dbl_out  <  infile
   dbl_out  >  outfile
   dbl_out  <  infile  >  outfile

*/
```

38