Day07 A

Young W. Lim

2017-10-07 Sat

Outline

Based on

"C How to Program",
Paul Deitel and Harvey Deitel

Program Modules in C

- to divide a large program into several smaller program *modules*
- manageable *modules*
- modules are *functions* in C
- a function is invoked by a *function call*
  - mentions the function by name
  - proivides the necessary information (arguments)
- *information hiding*
  - hides detailed information
  - provides minimal information

Function Definition

```
return-value-type function-name (parameter-list)
{
    definitions
    statements
}
```

- return-value-type
  - the type of the value returned to the calling function
  - when no return value, use void
- parameter list
  - a comma separated list containing
    the definitions of the variables
    that will be passed to the function
  - when no argument, use void

Function Prototype

- declares the function's return type
- declares the number, types, and order of the parameters that the function expects to receive

- the *function prototypes* enable the compiler to verify that functions are called correctly
- the variable (parameter) names in a function prototype are ignored
- implicit type conversion

Function Call

- the function's name (a comma separated list of arguments)
- the arguments passed to a function should match
  in number, type, and order
  with the parameters in the function definition
- each argument of a function
  - a constant (10, 3.14, . . .)
  - a variable (i, x, . . .)
  - an expression (10%i, 2*3.14*x, . . .)

Function Arguments and Parameters

- function definition
  ```
  int func(int x, float y, char *z) { ... }
  ```
- function prototype
  ```
  int func(int x, float y, char *z) ;
  int func(int,   float,   char * ) ;
  ```
- function call
  ```
  S = func(10, 3.14, 'A');
  func(i, x, &ch);
  ```
- function parameters : x, y, z
- function arguments : 10, 3.14, 'A' / i, x, ch

Function Return

- *control* is transferred from the point of invocation to the called function
  - the statements of the called functions are executed
  - at the end, the control *returns* to the caller
- a called function *return control* to the caller
- when control is returned without any value
  - when the function ending right brace } is reached
  - or by executing return statement
- when control is returned with a value
  - by executing return expression

Function Header

- Each standard library has a corresponding header
  - contains the prototypes of all the functions in that library
  - contains definitions of various symbolic constants
- We can create and include our own headers
- `<math.h>` header file

Information Hiding

- a *local variable* is known only in a function <u>definition</u>
- other functions are not allowed to know
  the names of a function's *local variables*
- other functions are not allowed to know
  the *implementation details* (defintions) of any other function

Passing Arguements by value and by reference

- when an argument is passed by value,
    - the 'value' argument is <u>copied</u> into a corresponding parameter variable
    - changes to the copy in the called function do not affect the original variable.
- when an argument is passed by reference,
    - the 'address' argument is <u>copied</u> into a corresponding parameter *pointer* variable
    - can change the original variable through its address

call-by-value and call-by-reference

- All calls in C are call-by-value
  - passing arguments by value : value copying is involved
  - passing arguemtns by reference : address copying is involved
- but it is possible to simulate call-by-reference
  by using & and $*$ operators

Math Library Functions

basic functions	abs, fabs, div, fmod, fmax, fmin, . . .
exponential functions	exp, log, log2, log10, . . .
power functions	sqrt, pow, . . .
trigonometric functions	sin, cos, tan, asin, acos, atan, atan2
hyberbolic functions	sinh, cosh, tanh, asinh, acosh, atanh
error and gamma functions	erf, erfc, lgamma, tgamma
nearest integer functions	ceil, floor, trunc, round, rint, . . .
floating point functions	frexp, . . .
classfication functions	isinf, isnormal, . . .