ECE20019 Open Source Software, Spring 2018

Function

Programming in C

Functions

- A function is a piece of code that receives a list of values as input and returns one value
  - identified by name and argument types
  - has an independent scope

- Example

```
int getlenth(char * s) {
  int i = 0 ;
  for ( ; s != 0x0 ; s++) i++ ;
  return i ;
}

int main() {
  char * s1, *s2 ;
  ...
  if (getlenth(s1) < getlenth(s2)) {
    ...
```

- *function declaration*
- *function definition*
- *function name*
- *arguments*
- *return type*
- *call site*

What Is Function For?

- Functions break large code into smaller pieces
  - function hides the details of a caller from a callee (and vice versa), which reduces complexity
- Function allows programmers to abstract code into high-level operations
  - functions reduce redundancy in code
  - function allows programmers to write repeated executions of code, such as recursion
- Functions role as interfaces between two modules
  - the interface of a library in Unix (i.e., API) is a list of functions
  - e.g., main function

Function Declaration and Definition

- A function is declared as a triple of a return type, a function name and a list of argument types
  - a function may receive no argument
  - a function may return no value
  - there can be multiple function declarations sharing the same function name while having different lists of argument types (i.e., overloading)
  - a function may receive an arbitrary number of arguments

- A function is defined with a code block
  - each argument must be bonded with a specific variable name
  - the code block must return a value if the function has a return value

Execution Model

```
01  int get_lenth(char * s) {
02    int i = 0 ;
03    for ( ; s != 0x0 ; s++)
04      i++ ;
05    return i ;
06  }

07  int main() {
08    int i, max_lenth ;
09    char s[8][128] ;
10
11    for (i = 0 ; i < 8 ; i++)
12      scanf("%s", s[i]) ;
13
14    max_lenth = 0 ;
15    for (i = 0 ; i < 8 ; i++) {
16      int r ;
17      r = get_length(s[i]) ;
18      if (r > max_length)
19        max_lenth = r ;
20    }
21    printf("%d", r) ;
22  }
```

Passing Arguments

- Call by value
  - the value given as an argument is copied to a new variable at a function call
- Example

```
01  struct node {
02     struct node * next ;
03  } ;

04  struct node header ;

05  void add(struct node * header, struct node * e) {
06     e->next = header->next ;
07     header->next = e ;
08  }
```

Recursion

- Recursion is to define a solution of a problem with the solutions of the sub-problems
  - a sub-problem shapes in the same form as the original problem, yet having a smaller input
  - definition
    - base case
    - recursion step

- A function is recursive when it calls itself in its body
  - recursion allows a finite logic to solve a problem with an arbitrary size of input

Ex. Enumerating Combinations

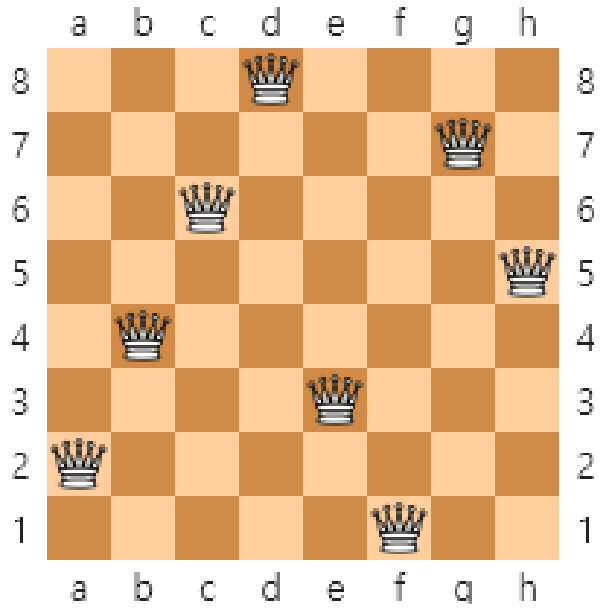
- Write a program that receives a list of integers and prints out all combinations of them
  - E.g., when {1, 2, 3} is given, the program should print outs:

    {}, {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, and {1, 2, 3}

Exercise 4

- Write a program that finds two placements of 8 Queens on 8-by 8 chessboard such that no Queen threatens each other
- Example

```
$ ./a.out
0 0 0 1 0 0 0 0
0 0 0 0 0 0 1 0
0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 0 1 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0
```



Exercise 5

- An arithmetic expression is one of two cases:
  - an integer
  - （ *exp op exp* ）
    - *expr* is an arithmetic expression
    - *op* is either +, −, *, or
- Write a program that reads an arithmetic expression and prints out the evaluation result

```
$./a.out
((1 + (2 * 3)) - (2 + 3))
2
$
```