Chapter 1. An Overview of C

Byoung-Tak Zhang
TA: Hanock Kwak

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National University

http://bi.snu.ac.kr
Algorithmic Thinking

- Computer is very diligent, but not so smart
- Computer must be told in detail what to do
  - With understandable codes to computer for all possible cases.
- **Algorithmic Thinking**
  - Algorithms = Recipes
Programming Languages

- **Algorithms**: Developed by people

Programming Languages

- High-level languages
- Assembly languages
- Machine languages

- **Computers**: Execute algorithms
How to Learn Programming

- Learn by doing
  - Do exercises/practices.
  - Lectures will give you basic tools only.

- In the lectures, you will learn:
  - Language syntax
  - Algorithmic thinking
  - Ideas

- Read “An Overview of C” & Try by yourself
Warning!!

- Lectures
  - seem easy

- Textbook: An Overview of C
  - seems that you understand well

- Programming assignments
  - more difficult than it seems

- Expect many bugs in your programs
  - Programming maturity comes with p.r.a.c.t.i.c.e!!
C Programming Language

- Born in the early 1970s with UNIX
- C is
  - Small
    - Fewer keywords
  - Portable
    - Code written on one machine easily moved to another
  - Terse
    - A very powerful set of operators
    - Able to access the machine in the bit level
  - Widely used
    - The basis for C++ and Java
C Programming Language

- Criticism
  - Complicated syntax
  - No automatic array bounds checking

- Nevertheless, C is elegant language
  - No straitjacket on the programmer’s access to the machine
  - Powerful operators
Hello World 1/3

- Create a C source file
  - http://www.tutorialspoint.com/compile_c_online.php

```c
#include <stdio.h>

int main()
{
    printf("Hello, World!\n");
    return 0;
}
```
Compilation & Linking

- Compile the source.
- Following command makes executable file (main) directly from the source files.
- It also do linking process after the compilation.

```
sh-4.3# gcc -o main *.c
sh-4.3#
```
Hello World 3/3

- **Execution**
  - Execute the program.

```
sh-4.3# gcc -o main *.c
sh-4.3# main
Hello, World!
sh-4.3#
```
Compilation & Linking

- **Compile**
  - Convert source files to object files.
  - Object file
    - It contains machine codes.
    - These object files are not executable yet, since they need to be combined and include other external codes (library).

- **Linking**
  - Combine **object files** and other **external codes** (library) to a single executable file.
Figure 1: The build process

- C Language Source File (main.c)
- C Language Header File

Compiler

Object File (main.o)

Linker

Library (C Standard Library, etc.)

Execution Program File (sample)
#include <stdio.h>

int main(void)
{
    printf("from sea to shining C\n");
    return 0;
}

from sea to shining C
#include <stdio.h>

- **Preprocessor**
  - It is built in the C compiler.
  - It manipulates source codes before the compilation.
  - They are also called as macro.

- **#include <[header file name]>**
  - It includes a copy of the header file into the source code.
  - For example, ‘#include <stdio.h>’ includes `stdio.h` file.
  - `stdio.h` is a standard library containing declaration of input/output functions.
int main(void)
{

}
printf()
- A function that prints on the screen
- It’s definition is in the header file `stdio.h`

“from sea to shinning C

- “... “: string constant in C
- \n: a single character called newline

printf(“from sea to shinning C

- Prints from sea to shinning C
Program Output

return 0;

- A return statement of the main()
- Causes the value zero to be returned to the operating system
- A value returned by the main() indicates how the program is finished.
- Normally zero means that the program is finished well.
```
#include <stdio.h>

int main(void)
{
    printf("from sea to ");
    printf("shining C");
    printf("\n");
    return 0;
}
```

```
#include <stdio.h>

int main(void)
{
    printf("from sea\n");
    printf("to shining\nC\n");
    return 0;
}
```
Errors in Source Codes

```c
#include <stdio.h>

int main(void)
{
    printf("from sea to shining C\n");
    return 0;
}
```
Errors in Source Codes

- Compilation failed
  - ‘return 0;’ is incorrect C language grammar.

```
sh-4.3# gcc -o main *.c
main.c: In function 'main':
main.c:6:5: error: 'return' undeclared (first use in this function)
  return 0;
  ^
main.c:6:5: note: each undeclared identifier is reported only once for each function it appears in
main.c:6:12: error: expected ';' before numeric constant
  return 0;
  ^
sh-4.3# 
```
/*the distance of a marathon in kilometers*/
#include <stdio.h>
int main(void)
{
    int miles, yards;
    float kilometers;

    miles = 26;
    yards = 385;
    kilometers = 1.609 * (miles + yards / 1760.0);
    printf("A marathon is \%f kilometers.\n\n", kilometers);
    return 0;
}
Variable, Expression, Assignment

/*the distance of a marathon in kilometers*/

- /* ... */
  - comment (used for documentation or memo)
  - ignored by the compiler
int miles, yards;
- declaration of the variables miles and yards of type integer (int)
- Declarations and statements end with a semicolon.

float kilometers;
- float
  - real value type
  - shortened version of ‘floating point number’
- declaration of a variable kilometers of type float
Variable, Expression, Assignment

$miles = 26;$
- assignment statement
- 26 is assigned to a variable miles

$kilometers = 1.609 * (miles + yards / 1760.0);$  
- The value of the expression on the right side of the equal sign is assigned to a variable kilometers
Variable, Expression, Assignment

printf("\nA marathon is %f kilometers.\n\n", kilometers);

- %f
  - format, conversion specification
  - matched with the remaining argument, the variable kilometers

printf("\nA marathon is %f kilometers.\n\n", kilometers);
Variable, Expression, Assignment

Terminal

sh-4.3# gcc -o main *.c
sh-4.3# main

A marathon is 42.185970 kilometers.

sh-4.3#
#include <stdio.h>

int main(void)
{
    int  a, b;

    ......

    a = 1;
    if ( b == 3 )
        a = 5;
    printf("%d", a);
    return 0;
}
Flow of Control

if (expr)
    statement
- If \textit{expr} is nonzero(true), then \textit{statement} is executed.

if (b==3)
    a = 5;
- \texttt{==}: equal operator
  - \texttt{a==b} is one if \texttt{a} and \texttt{b} are same, otherwise zero.
  - for example, \texttt{3==3} is one, and \texttt{2==3} is zero
  - \texttt{a = 5}; will be executed only if \texttt{b==3} is one(true); that is, \texttt{b} is 3.
Flow of Control

```
#include <stdio.h>
int main(void)
{
    int a, b;
    b = 3;
    a = 1;
    if ( b == 3 )
        a = 5;
    printf("%d", a);
    return 0;
}
```
Flow of Control

```java
if (a == 3)
{
    b = 5;
    c = 7;
}
```

- **Compound statement** `{...}`
  - a group of statements surrounded by braces
  - a statement, itself
if (expr)
    statement1
else
    statement2

if (a == 3)
{
    b = 5;
    c = 7;
}
else
{
    a = 10;
    b = a + c;
}
Flow of Control

```c
#include <stdio.h>
int main(void)
{
    int i = 1, sum = 0;

    while ( i <= 5 )
    {
        sum = sum + i;
        ++i;
    }
    printf("sum = %d\n", sum);
    return 0;
}
```
while (expr)

statement

expr is true

expr is true

YES

NO

statement
Flow of Control

while (i <= 5)
{
    sum = sum + i;
    ++i;
}

- **sum** is added by *i*, until *i* is less than or equal to 5
- **++i**
  - ++ : increment operator
  - i = i + 1
```c
#include <stdio.h>
int main(void)
{
    int i = 1, sum = 0;

    while (i <= 5)
    {
        sum = sum + i;
        ++i;
    }

    printf("sum = %d\n", sum);
    return 0;
}
```

1+2+3+4+5

sum = 15
C Program is ...

- A sequence of **FUNCTIONS**
  - main() function executed first
- A **FUNCTION** consists of:
  - Declarations
  - Statements
- **Declaration**: variable names and their types
  - `int miles;`
- **Statement**: data processing or control
  - `miles = 26;`
  - `if (b == 3) { ...};`
  - `printf(...);`