AN OVERVIEW OF C
Algorithmic Thinking

- Very diligent
- But, not so smart

- Must be told in detail what to do
  - understandable to computer
  - for all possible cases

- Algorithmic Thinking
  - Algorithms == Recipes
Programming Languages

- **Algorithms**: Developed by people
  - 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
  - A Book on C
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 practice!!
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
  - Multiple use of such symbols as * and =
    - **, ==

- Nevertheless, C is an elegant language
  - No straitjacket on the programmer’s access to the machine
  - Powerful operators
1. **Create a C source file**
   - use a text editor
     - Vi, text editor, Atom, ...

```c
#include <stdio.h>

int main(void)
{
    printf( "Hello world!\n" );
    return 0;
}
```
Hello world 2/3

2. Compile
   1. Convert source codes to object codes
   2. Compiler does the job
3. **Linking**
   - Convert object codes to executable file
   - Linker does the job

4. **Debugging**
   - Fix the bugs in the source codes
   - Debugger does the job

5. **Run or Execute**
From Source to Executable
```c
#include <stdio.h>

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

from sea to shining C

Source file: sea.c
Program Output

#include <stdio.h>

- Preprocessor
  - built into the C compiler
  - Lines beginning with #: communicate with the preprocessor

- #include
  - Preprocessor includes a copy of the header file stdio.h
  - stdio.h
    - provided by the C system
    - Declaration of standard input/output functions, e.g., printf()
int main(void)

- The 1\textsuperscript{st} line of the function definition for main()
- int, void
  - keywords, or reserved words
  - Special meanings to the compiler

int main(void)
{
- Every program has a function named \texttt{main()}
- void, no argument / return an \texttt{int} value
- \{ ... \}, the body of a function definition
**printf()**

- A function that prints on the screen
- information in the header file `stdio.h`

"from sea to shinning C\n"

- "\": string constant in C
- \n: a single character called `newline`

`printf("from sea to shinning C\n");`

- statement: end with a semicolon
return 0;

- A return statement
- causes the value zero to be returned to the operating system

} 

- The right brace matches the left brace
- ending the function definition for main()
Compiling

- Convert source file to objective file
  - sea.c to sea.o (or sea.obj)

- Object file
  - A file with expressions that computers can understand

- When compiling fails?
  - Something wrong with source file ...
    - Expressions with wrong C grammar
Errors in Source File (ex)

```c
#include <stdio.h>

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

- return 0;
- incorrect C language grammar
- compiler fails to make an obj file and returns an error.
- debugging
  - change "return 0;" to "return 0;"
Errors in Source File (ex)

D:\> bcc hello.c
Borland C++ 5.5.1 for Win32 Copyright (c) 1993, 2000 Borland
hello.c:
Error E2451 hello.c 6: Undefined symbol 'return' in function main
Error E2379 hello.c 6: Statement missing ; in function main
Warning W8070 hello.c 7: Function should return a value in function main
*** 2 errors in Compile ***
D:\>
Linking and Running a Program

- Linking
  - The process to make an executable program out of objective file(s)
    - sea.o (or sea.obj) → a.out (sea.exe)

- Run a program
  - type "a.out" or "sea"
    - computer prints "from see to shining C"
```c
#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;
}

from sea to shining C

from sea
to shining
C
/*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, Expressions & Assignment

/*the distance of a marathon in kilometers*/

/* ... */

- comment
- 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
  - a keyword, real value
- declaration of the variable kilometers of type float
miles = 26;
yards = 385;

- **Assignment statement**
- **=: assignment operator**

kilometers = 1.609 * (miles + yards / 1760.0);

- **Assignment statement**
- The value of the expression on the right side of the equal sign is assigned to the variable **kilometers**
printf("\nA marathon is %f kilometers.\n\n", kilometers);

- Control string
- %f: format, conversion specification
  - Matched with the remaining argument, the variable kilometers
Variable, Expressions & Assignment

/*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;
}

A marathon is 42.195970 kilometers.
1.609, 1760.0

- A decimal point indicates that a floating-point constant rather than an integer constant.
- Three floating types: float, double, long double.
- Floating-point constants are automatically of type double.
Variable, Expressions & Assignment

Expression

- On the right side of assignment operators
- Constants, variables, or combinations of operators with variables and constants

  e.g) yards = 385;

  kilometers = 1.609 * (miles + yards / 1760.0);

Evaluation of Expression

- Conversion rule
  - Division of two integers results in an integer values. 7/2 is 3
  - A double divided by an integer
    - Integer is automatically converted to double
    - 7.0/2 is 3.5

  kilometers = 1.609 * (miles + yards / 1760.0);   bug!!!
Flow of Control

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

```java
if (expr)
    statement
```
- If `expr` is nonzero (true), then the `statement` is executed;
- otherwise, it is skipped

```java
if (b==3)
    a = 5;
```
- `==` : equal to operator
- `b==3` : logical expression: either the integer value 1 (true) or 0 (false)
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;
}
```

```
#include <stdio.h>
int main(void)
{
    int a, b;
    b = 2;
    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 statement surrounded by braces
- A statement, itself
Flow of Control

if (expr)
    statement1
else
    statement2

if (cnt == 0)
{
    a = 2;
    b = 3;
    c = 5;
}
else
{
    a = -2;
    b = -3;
    c = -5;
}
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;
}
```
Flow of Control

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

- If `expr` is true, the compound statement is executed,
- and control is passed back to the beginning of the `while` loop for the process to start over again
- The `while` loop is repeatedly executed until the test fails

++i;
- `++` : increment operator
- `i = i + 1;`
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;
}
```

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(...);