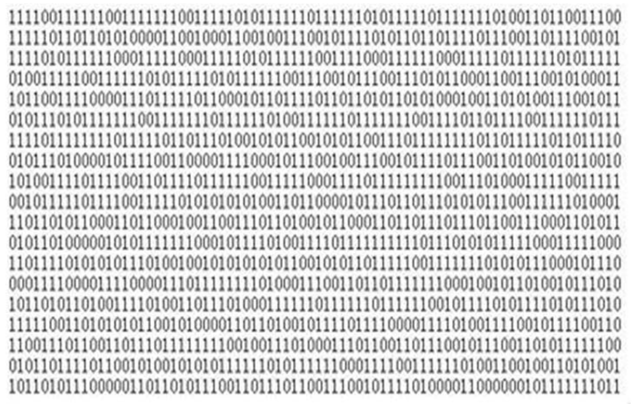
Chapter 1
Programming Fundamentals

- Programming Languages
- Software Engineering
- C and ANSI C
- Program Development
- Exploration
- Simple Output Formatting

1.1 Programming Languages

- We need to communicate with the computer
- A computer can only understand "Machine Language"
- Machine Language
  - "Machine language" consists of instructions in binary code, that is, a combination of 0 and 1 codes. Very difficult for human to write
- Thus programs are written in other languages and translated into machine language

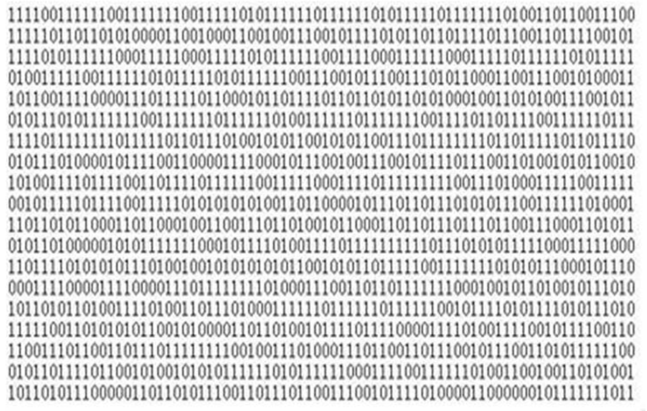


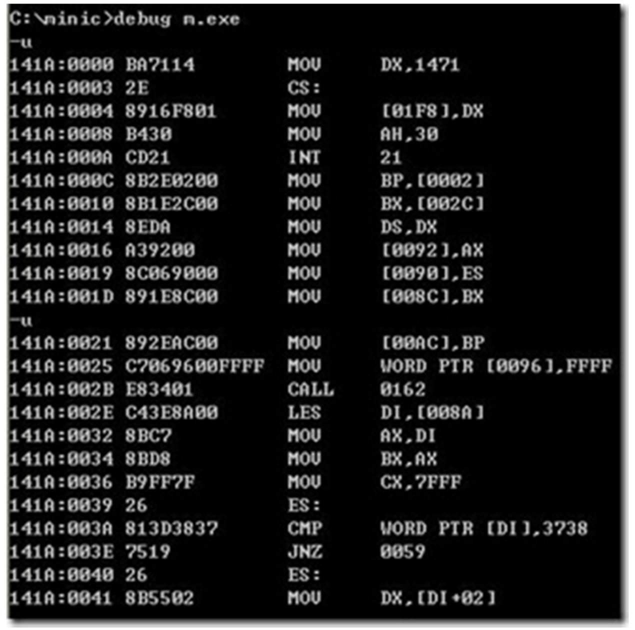
Programming Languages

- Two favours
  - Low level / Assembly Language
  - High level language
- Thus programs are written in other languages and translated into machine language

Assembly Language

- All instructions have a one-to-one mapping with machine language counterpart
- E.g. machine code “10010101” in assembly language will be “ADD”
- Assembly language instructions are not in binary code but in English words – easier to memorize & use
- Problems
  - Programmer needs complete understanding of computer hardware
  - Substantial code needed just to do some simple tasks, e.g. print a message





```
C:\minic>debug m.exe
-u
141A:0000 BA7114        MOV     DX,1471
141A:0003 2E            CS:
141A:0004 8916F801      MOV     [01F8],DX
141A:0008 B430          MOV     AH,30
141A:000A CD21          INT     21
141A:000C 8B2E0200      MOV     BP,[0002]
141A:0010 8B1E2C00      MOV     BX,[002C]
141A:0014 8EDA          MOV     DS,DX
141A:0016 A39200        MOV     [0092],AX
141A:0019 8C069000      MOV     [0090],ES
141A:001D 891E8C00      MOV     [008C],BX
-u
141A:0021 892EAC00      MOV     [00AC],BP
141A:0025 C7069600FFFF  MOV     WORD PTR [0096],FFFF
141A:002B E83401        CALL    0162
141A:002E C43E8A00      LES     DI,[008A]
141A:0032 8BC7          MOV     AX,DI
141A:0034 8BD8          MOV     BX,AX
141A:0036 B9FF7F        MOV     CX,7FFF
141A:0039 26            ES:
141A:003A 813D3837      CMP     WORD PTR [DI],3738
141A:003E 7519          JNZ     0059
141A:0040 26            ES:
141A:0041 8B5502        MOV     DX,[DI+02]
```

High Level Languages

- Simplify the commands needed to be written by human, e.g., **print ("a message");**
- Write programs with far less concern about the internal design of the machine
- Can be broken down into four types:
  - Procedural 过程式/程序式 (or imperative 命令式/指令式)
  - Functional 函数式
  - Declarative 声明式
  - Object oriented 面向对象
- C language is *procedural* type – requires the programmer to lay out a ***procedure*** for solving a problem
- C is a *subset* of C++ : Everything you learn from this text about C can be applied to C++
- Discuss:
  - **How many type of High Level Language you have known?**

Summary of some high level languages

Language	**Type**	**Year developed**
Fortran	Procedural	Mid 1950s
Basic	Procedural	Mid 1960s
Lisp	Functional	Late 1950s
Prolog	Declarative	Early 1970s
Smalltalk	Object oriented	Mid 1970s
Pascal	Procedural	Early 1970s
C	**Procedural**	**Mid 1970s**
C++	**Object oriented**	**Mid 1980s**
Java	Object oriented	Mid 1990s

Language Translators

- Language translators are **programs** that create machine language instructions (*object code*) from instructions written in assembly/high level language
- Three favours
  - *Assemblers* 汇编器: Convert programs (in assembly language) to object code
  - *Compilers* 编译器: Taking an *entire* program (in high level language) and converting it to machine instructions
  - *Interpreters* 解释器: Translate and execute (high level language) instructions *one after another*.

high level language

```
main()
{
int x;
x=1;
}
```

⇨

Translator

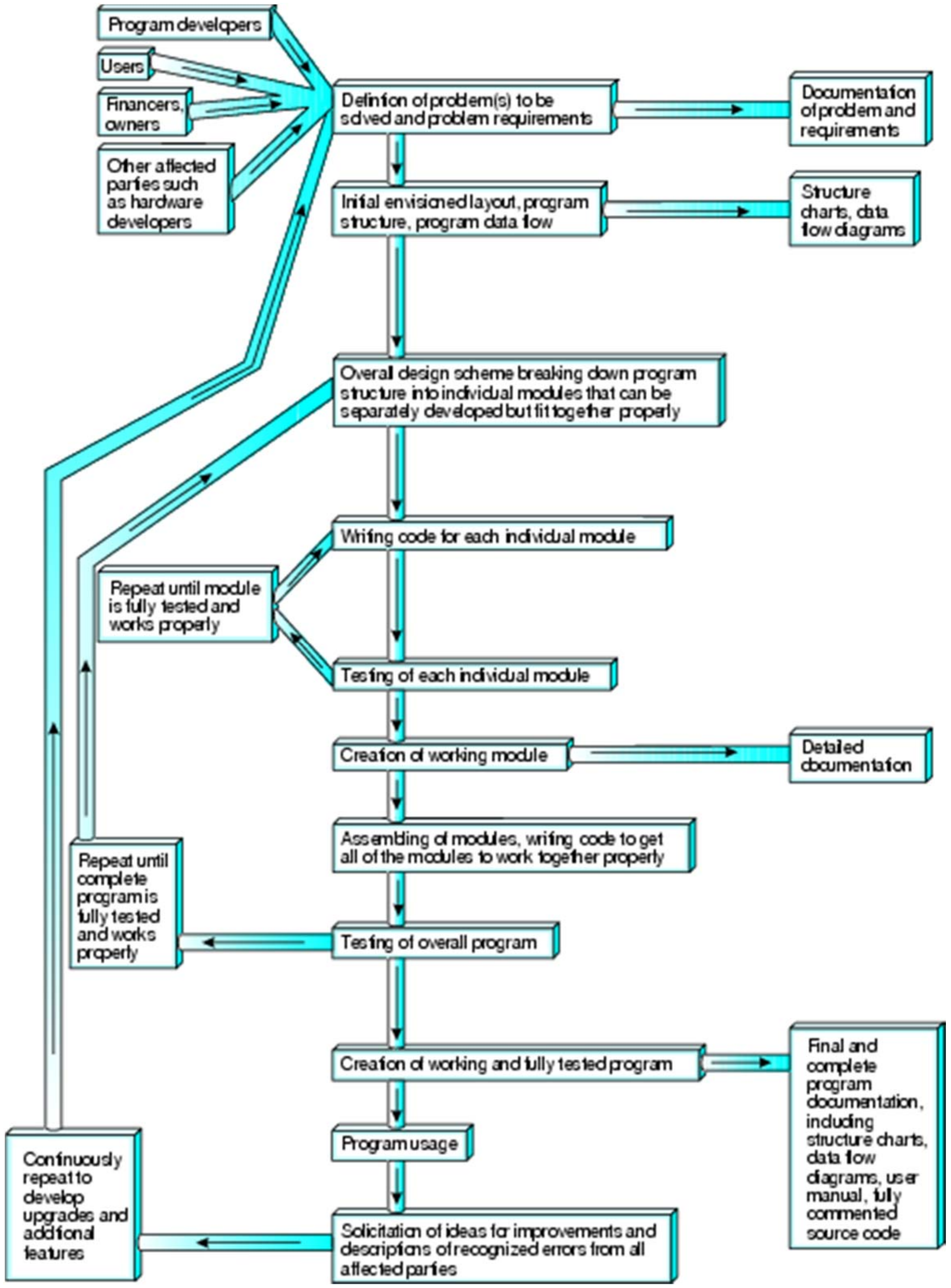
⇨

object code

```
MOVF   id3,R2
MULF    #10.0, R2
MOVF   id2,R1
ADDF    R1, R2
MOV     R1, id1
```

1. 2 Software Engineering

- The process of software development
- Should be thoroughly thought out, planned, constructed, and tested
  - Define function of the software
  - Develop sketch of the layout
  - Input from users, owners, programmers
  - Design of individual components is addressed
  - Planning modifications and assemble the software and test for functionality
  - Comprehensively tested and modified as necessary
  - Documentation about the software is maintained
- Discuss：
  - **How to copy a page of letters to a blank paper**



Program developers
Users
Financers, owners
Other affected parties such as hardware developers
Definition of problem(s) to be solved and problem requirements
Documentation of problem and requirements
Initial envisioned layout, program structure, program data flow
Structure charts, data flow diagrams
Overall design scheme breaking down program structure into individual modules that can be separately developed but fit together properly
Writing code for each individual module
Repeat until module is fully tested and works properly
Testing of each individual module
Creation of working module
Detailed documentation
Assembling of modules, writing code to get all of the modules to work together properly
Repeat until complete program is fully tested and works properly
Testing of overall program
Creation of working and fully tested program
Final and complete program documentation, including structure charts, data flow diagrams, user manual, fully commented source code
Program usage
Continuously repeat to develop upgrades and additional features
Solicitation of ideas for improvements and descriptions of recognized errors from all affected parties

Top-down Modular Design

- Begins by defining the main module of the software
- Then sub-modules are developed
- Each module is less complex than the whole (upper level one)
- Modules are called functions in C
- Can be divided into
  - **Library functions**:
    - Already included with C language
  - **User-defined functions**:
    - Custom-made by C programmer

1.3 C and ANSI C

- Developed in early 1970s at Bell Laboratories by Dennis Ritchie
- Highly *portable(可移植的)*, i.e., machine independent
- In 1989, American National Standards Institute (Committee X3J11) approved a version of C – **ANSI C**
- In 1990, **ISO (**International Organization for Standardization**) C** was adopted.
- In 1999, a new version ISO/IEC 9899:1999 C is being introduced (**C99**)
- In 2007, work began on another revision of the C standard, informally called "C1X" until its official publication on 2011-12-08 – **C11**
- In this text, we follow the ANSI C standard for broader supports

Program Development

- Objective:
  - Create an executable file *(.exe*)
- A modern C programming development environment will:
  - Allowing user to edit text (create C source code)
  - *Preprocessing* source code
  - *Compiling* source code and indicating any possible errors
  - *Linking* object code with library (other object codes)

1.4 Using Bits to Represent …

- Home Reading (page 11)

1.5 About this Textbook

- Covered in the first lecture

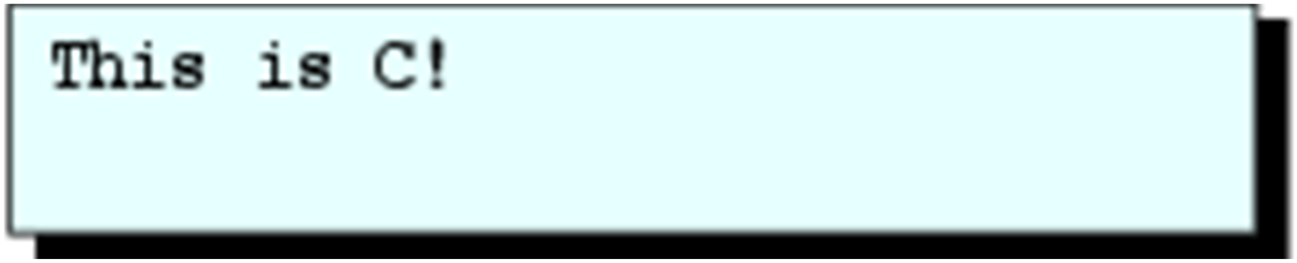
1.6 Basic Structure

- Topics to learn
  - Writing a simple but *complete* C program
  - Using the *printf*( ) function to display text on the screen
  - Structure of a simple C program
  - Basic rules for writing a C program

Program

```
Line No            Code
01   /*L1_1.C - In this book, the source for Lesson x_y is Lx_y.c */
02   #include <stdio.h>  /* This is an include directive */
03   void main(void)
04   {/*The purpose of this program is to print one
05          statement to the screen */
06     printf("This is C!");
07   }
```

Output

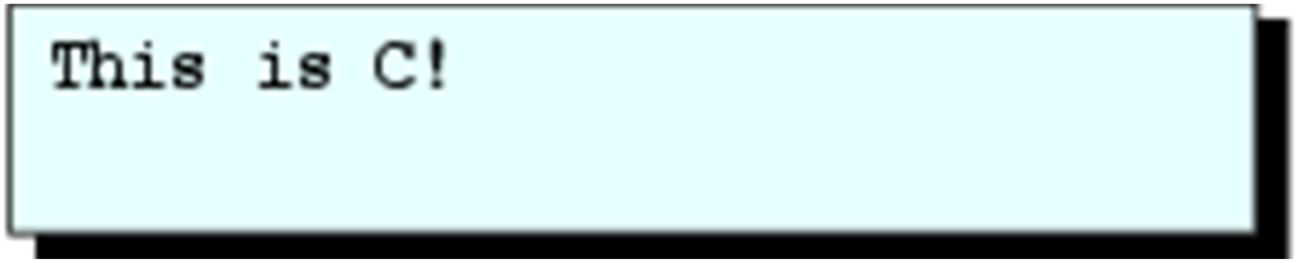


```
This is C!
```

Program

```
Line No          Code
01   /*L1_1.C - In this book, the source for Lesson x_y is Lx_y.c */
02   #include <stdio.h>  /* This is an include directive */
03   void main(void)
04   {/*The purpose of this program is to print one
05        statement to the screen */
06     printf("This is C!");
07   }
```

Comments : Notes describing a particular portion of your program

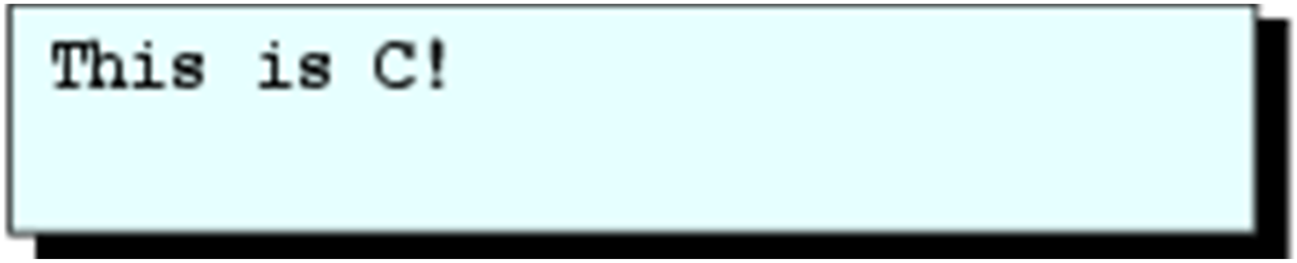


```
This is C!
```

Program

```
Line No             Code
01   /*L1_1.C - In this book, the source for Lesson x_y is Lx_y.c */
02   #include <stdio.h>   /* This is an include directive */
03   void main(void)
04   {/*The purpose of this program is to print one
05        statement to the screen */
06     printf("This is C!");
07   }
```

Preprocessor directives: contain info about the library function *printf*



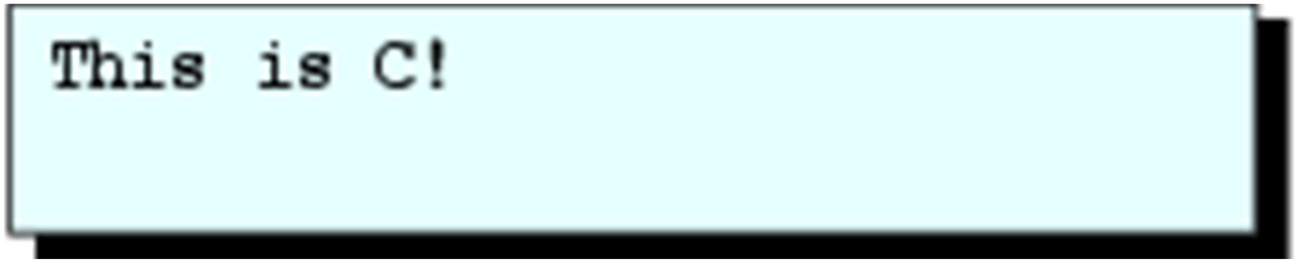
```
This is C!
```

Program

Line No   Code

```
01   /*L1_1.C - In this book, the source for Lesson x_y is Lx_y.c */
02   #include <stdio.h>  /* This is an include directive */
03   void main(void)
04   {/*The purpose of this program is to print one
05          statement to the screen */
06     printf("This is C!");
07   }
```

Entrance to a program
First function to be executed

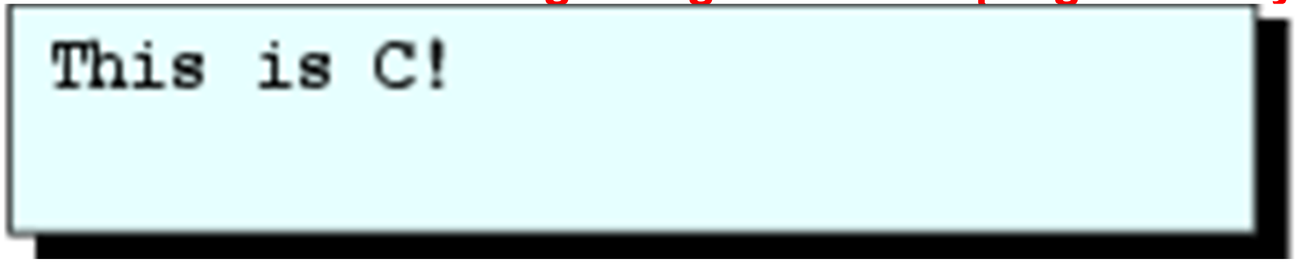


```
This is C!
```

Program

```
Line No     Code
01   /*L1_1.C - In this book, the source for Lesson x_y is Lx_y.c */
02   #include <stdio.h>   /* This is an include directive */
03   void main(void)
04   {/*The purpose of this program is to print one
05          statement to the screen */
06     printf("This is C!");
07   }
```

Braces: Mark the beginning & end of a program body



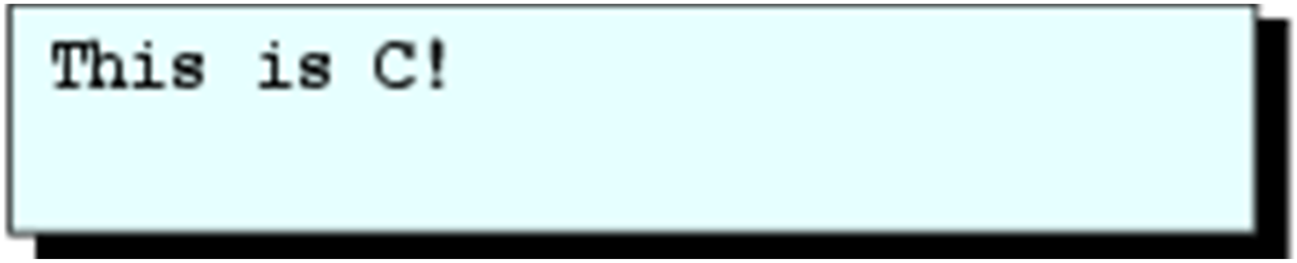
```
This is C!
```

Program

Line No Code

```
01   /*L1_1.C - In this book, the source for Lesson x_y is Lx_y.c */
02   #include <stdio.h>  /* This is an include directive */
03   void main(void)
04   {/*The purpose of this program is to print one
05        statement to the screen */
06     printf("This is C!");
07   }
```

Statements that instruct computer to print a message



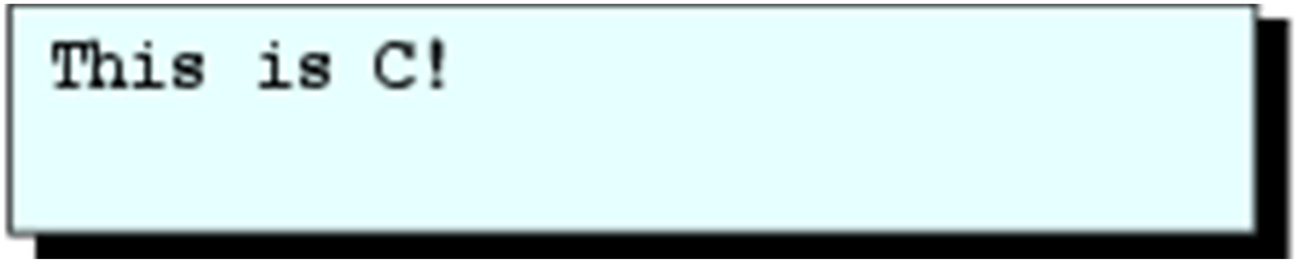
```
This is C!
```

Program

Line No   Code

```
01   /*L1_1.C - In this book, the source for Lesson x_y is Lx_y.c */
02   #include <stdio.h>   /* This is an include directive */
03   void main(void)
04   {/*The purpose of this program is to print one
05        statement to the screen */

06     printf("This is C!");
07   }
```

C statements must end with semicolon ;



```
This is C!
```

Further Exploration

- Q: Can we use both uppercase and lowercase letters to write C code? (case sensitive)

- A: Yes!
  - C language distinguishes between lower- and uppercase letters, i.e. ‘main’ is different from ‘Main’
  - Both *main* and *printf* must be written in lowercase letters

Further Exploration

- Q: Where are blank space(s) permitted in C code?
- A:
  - All C words should be written continuously, e.g.,
  - **“void ma   in(void)”** is not legal
  - But “**void main (void)** “ is okay
  - In general, it is acceptable to add blanks between words (we called it *tokens*) but not within

Further Exploration

- Q: Is it necessary to use different lines in writing code?
- A:
  - You have the freedom to write C code at any row or column you like
  - E.g.

```
#include <stdio.h>void main(void){printf("This is C!");}


or
#include<stdio.h>void
main(  void            )  {
   printf
(  "This is C!"  )   ;
are both valid
```

Basic Structure of C Program

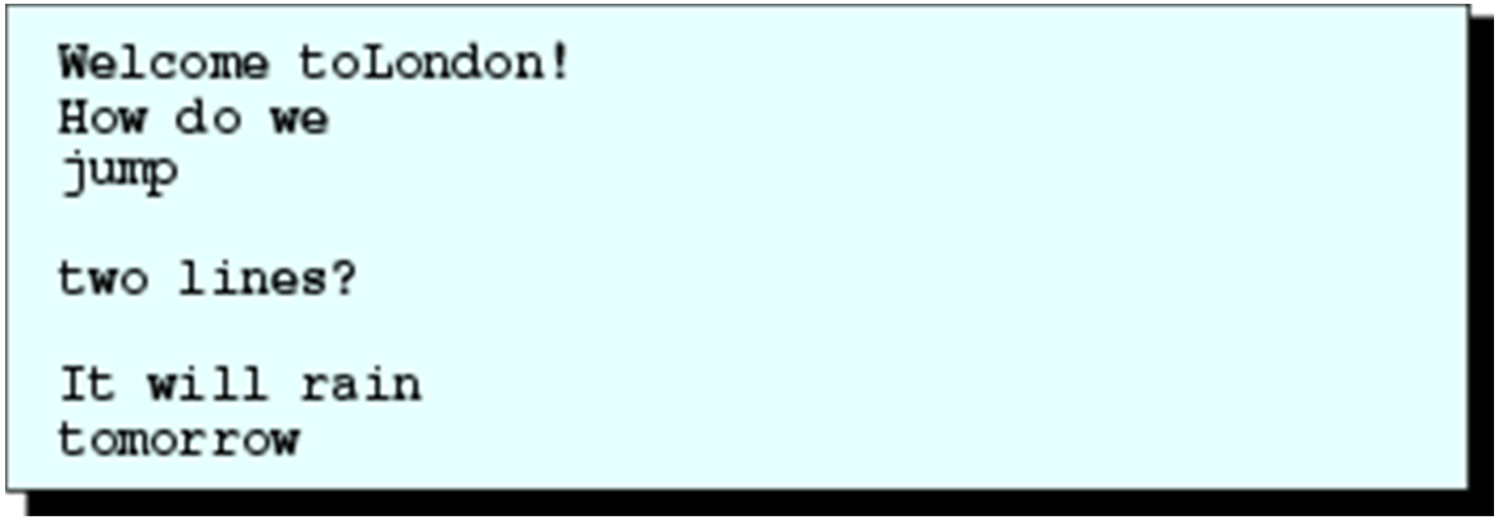
```
# preprocessing directives

void main(void)
{
  declaration statements;
  executable statements;
} /* any text, number, or character */
```

Home Reading: Further Exploration (page 24)

1.7 Formatting Output

```c
printf("Welcome to");
printf("London!");
printf("\nHow do we\njump\n\ntwo lines?\n");
printf("\n");
printf("It will rain\ntomorrow\n");
```

Output



```
Welcome toLondon!
How do we
jump

two lines?

It will rain
tomorrow
```

1.8 More Escape Sequences

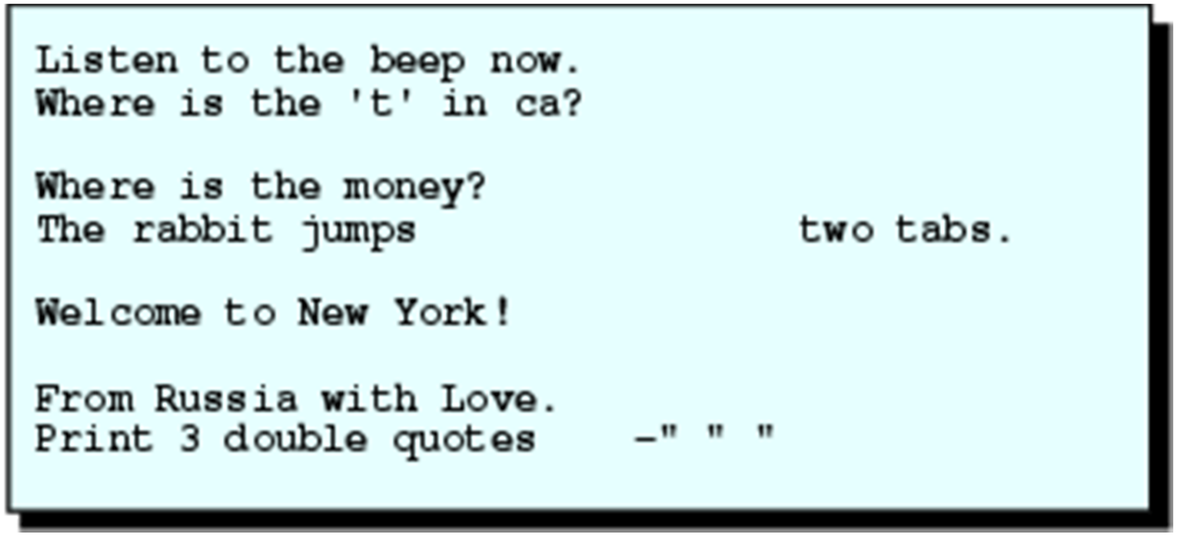
Character escape sequences [Book, pp. 32]

Escape Sequence	Meaning	Result
\0	Null character	Terminates a character string
\a	Alert/bell	Generates an audible or visible alert
\b	Backspace	Moves back one space on the current line
\f	Form feed	Moves to start of the next logical page
\n	**New line**	**Linefeeds to next line**
\r	**Carriage return**	**Moves to initial position of the current line**
\t	Horizontal tab	Moves to next horizontal tabulation position
\v	Vertical tab	Moves to next vertical tabulation position
\0ddd	Octal constant	integer constant of base 8 ddd digits 0-7
\xddd \Xddd	Hexadecimal constant	integer constant base 16, where ddd represents decimal digits, and a–f or A–F represent values of 10 through 15 respectively
\\	Backslash	Displays a backslash
\'	Single quote	Displays a single quote
\"	Double quote	Displays a double quote
\%	Percent	Displays a percent character

1.8 More Escape Sequences

```c
    printf("Listen to the beep now. \a");
    printf("\nWhere is the 't' in cat\b?\n\n");
    printf("I earned $50 \r Where is the money?\n");
    printf("The rabbit jumps \t\t two tabs.\n\n");
    printf("Welcome to \
New York!\n\n");
    printf("From "      "Russia \
with "      "Love.\n");
    printf("Print 3 double quotes   -\" \" \" \n");
```

Output



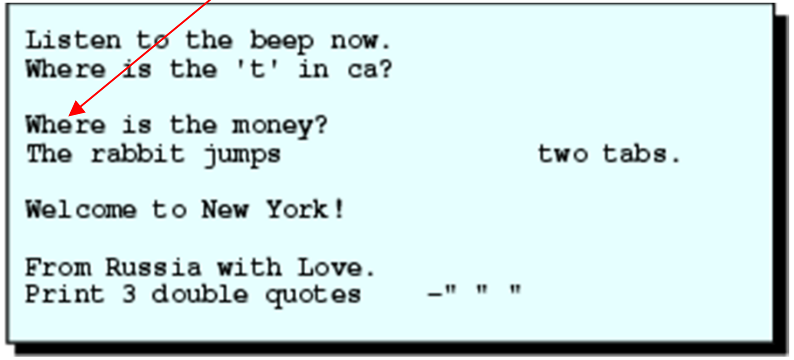
```
Listen to the beep now.
Where is the 't' in ca?

Where is the money?
The rabbit jumps                   two tabs.

Welcome to New York!

From Russia with Love.
Print 3 double quotes    -" " "
```

1.8 More Escape Sequences

Moving cursor to start column

```
Line
01
02
03 {
04     printf("Listen to the beep now. \a");
05     printf("\nWhere is the 't' in cat\b?\n\n");
06     printf("I earned $50 \r Where is the money?\n");
07     printf("The rabbit jumps \t\t two tabs.\n\n");
08     printf("Welcome to \
09 New York!\n\n");
10     printf("From "      "Russia \
11 with "      "Love.\n");
12     printf("Print 3 double quotes   -\" \" \" \n");
13 }
```

Output



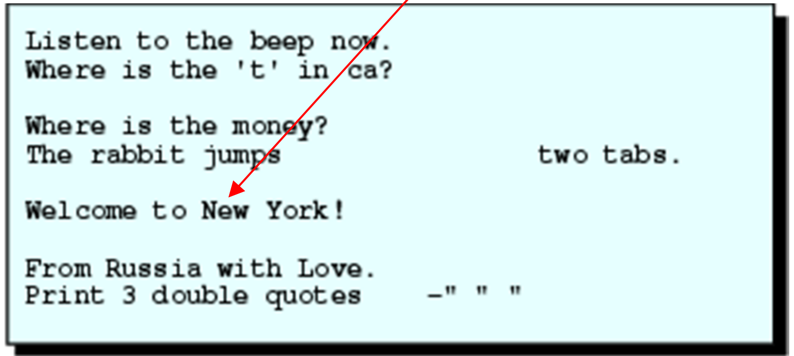
```
Listen to the beep now.
Where is the 't' in ca?

Where is the money?
The rabbit jumps                two tabs.

Welcome to New York!

From Russia with Love.
Print 3 double quotes    -" " "
```

1.8 More Escape Sequences

Concatenate two strings

```
Line
01
02
03 {
04      printf("Listen to the beep now. \a");
05      printf("\nWhere is the 't' in cat\b?\n\n");
06      printf("I earned $50 \r Where is the money?\n");
07      printf("The rabbit jumps \t\t two tabs.\n\n");
08      printf("Welcome to \
09 New York!\n\n");
10      printf("From "      "Russia \
11 with "      "Love.\n");
12      printf("Print 3 double quotes   -\" \" \" \n");
13 }
```

Output



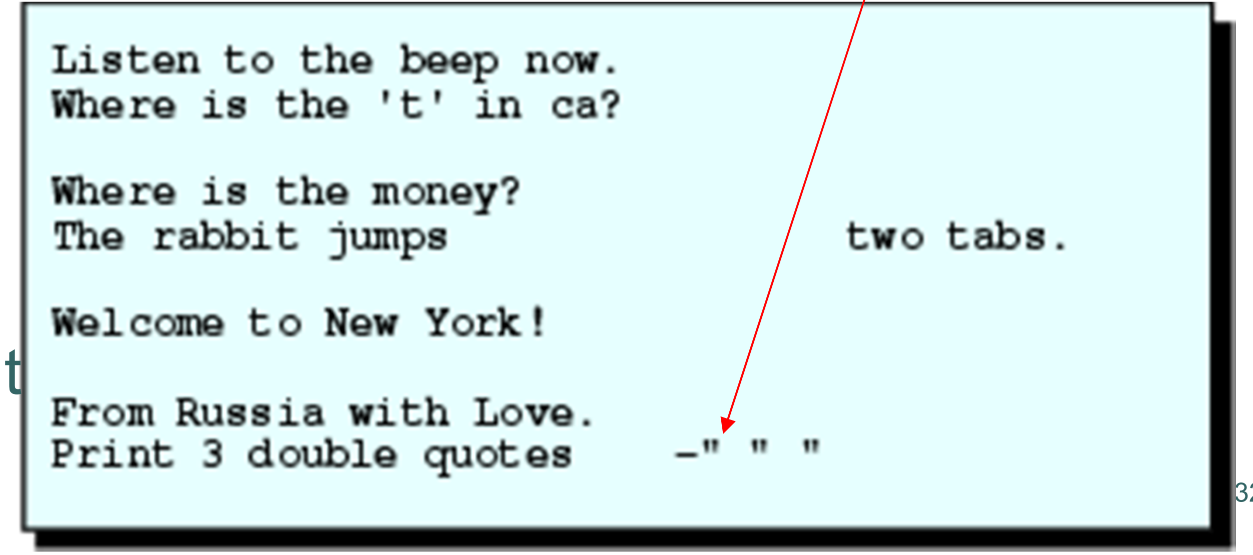
```
Listen to the beep now.
Where is the 't' in ca?

Where is the money?
The rabbit jumps                two tabs.

Welcome to New York!

From Russia with Love.
Print 3 double quotes    -" " "
```

1.8 More Escape Sequences

```
{
     printf("Listen to the beep now. \a");
     printf("\nWhere is the 't' in cat\b?\n\n");
     printf("I earned $50 \r Where is the money?\n");
     printf("The rabbit jumps \t\t two tabs.\n\n");
     printf("Welcome to \
New York!\n\n");
     printf("From "      "Russia \
with "      "Love.\n");
     printf("Print 3 double quotes   -\" \" \" \n");
}
```

Print double quotes

Output



```
Listen to the beep now.
Where is the 't' in ca?

Where is the money?
The rabbit jumps                  two tabs.

Welcome to New York!

From Russia with Love.
Print 3 double quotes    -" " "
```

1.9 Basic Debugging

- Syntax errors
- Run-time/semantic/smart errors
- Logic errors

- Home Reading (page 35)