Chapter 1 & 2

Introduction to C Language

Chapter 1 & 2 - Introduction to C Language

Outline

Chapter 1

1.1 The History of C
1.2 The C Standard Library
1.3 C++
1.4 The Basics of a typical C Program Development Environment

Chapter 2

2.1 A Simple C Program: Printing a Line of Text
2.2 Another Simple C Program: Adding Two Integers
2.3 Memory Concepts
2.4 Arithmetic in C
2.5 Decision Making: Equality and Relational Operators

C Reserved Keywords

- Keywords are the special words reserved for C.
- They cannot be used as identifiers or variable names.
- There are 32 keywords in standard C.

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

1.1 History of C

- C Language
  - Evolved by Dennis Ritchie from two previous programming languages, BCPL and B
  - Used to develop UNIX
  - Used to write modern operating systems
  - Hardware independent (portable)
    - The same C program may run on many different hardware platforms with or without small modifications
  - Standard created in 1989 (ANSI) , updated in 1999 (ISO)
  - The latest C language standard is C11

1.2 The C Standard Library

- C programs consist of modules called functions
  - A programmer can create his own functions
    - Advantage: the programmer knows exactly how it works
    - Disadvantage: time consuming
  - Programmers will often use the C library functions
    - Use these as building blocks
  - Avoid re-inventing the wheel
    - If a pre-made function exists, generally best to use it rather than write your own
    - Library functions carefully written, efficient, and portable

1.3 C++

- C++ Language
  - Superset of C developed by Bjarne Stroustrup at Bell Labs
  - Extends the C, and provides object-oriented capabilities
  - Because C++ includes C, it is best to master C,
    then learn C++

2.1 A Simple C Program: Printing Text

```c
/* Fig. 2.3: fig02_03.c
   Printing on one line with two printf statements */
#include <stdio.h>

int main() // Execution starts here
{
   printf( "Welcome " );
   printf( "to C!\n" );
}
```

Program Output

```
Welcome to C!
```

2.1 A Simple C Program: Printing Text

Comments

- Multi-line comments are written as /* ……..*/
- One-line comments are written as // ……..
- Usually // notation is preferred most of the time
- Comment texts are ignored by compiler
- Used to describe program

- `#include <stdio.h>`
  - Preprocessor directive
    - Tells computer to load contents of a certain file
  - `<stdio.h>` allows standard input/output operations

2.1 A Simple C Program: Printing Text

- `int main()`
  - C/C++ programs contain one or more functions, exactly one of which must be `main`
  - Parenthesis used to indicate a function
  - `int` means that `main` can "return" an integer value
  - Braces `{` and `}` indicate a block
    - The bodies of all functions must be contained in braces

- Right brace `}`
  - Indicates end of `main` has been reached

main's Return Type

Notice that `main` has an `int` return type.

- The optional return statement is a way to exit a function
- The return value of `main` is used to indicate whether the program executed correctly.
- In earlier versions of C, we'd explicitly place

  `return 0;`

  at the end of `main`, `0` indicates that a program ran successfully.
- The latest C standard indicates that main implicitly returns 0 if you to omit the preceding statement.

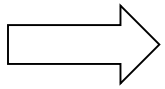
```c
#include <stdio.h>
int main()
{
   printf("Welcome to C\n");
   return 0; // This statement can be omitted
}
```

2.1 A Simple C Program: Printing Text

- `printf( "Welcome " );`
  `printf( "to C!\n" );`

  - Instructs computer to perform an action
    - Specifically, prints the string of characters within quotes (" ")
  - Each entire instruction is called a statement
    - All statements must end with a semicolon (;)
  - Escape character (\)
    - Indicates that printf should do something out of the ordinary
    - \n is the newline character

```
printf( "Welcome\nto\nC!\n" );
```



```
Welcome
to
C!
```

2.1 A Simple C Program: Printing Text

Escape Sequence	Description
`\n`	Newline. Position the cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the cursor to the next tab stop.
`\a`	Alert. Sound the system bell.
`\\`	Backslash. Insert a backslash character in a string.
`\"`	Double quote. Insert a double quote character in a string.

Fig. 2. Some common escape sequences.

Variable Declarations

- A variable is a RAM memory location to store a data value.

- For example, the definition (declaration)

```
int x;
```

  tells the compiler the type (integer) of variable x and instructs the compiler to reserve space in memory for the variable.

- But this definition does *not* cause any action—such as input, output, a calculation or a comparison—to occur when the program is executed.

Initialization of Variables

- Variables should normally be initialized to some value before being used in a program; otherwise a variable would include the previous value stored in the variable's memory location.

- An uninitialized variable contains a garbage value—the value last stored in the memory location reserved for that variable.

Program 1

```
#include <stdio.h>
int main()
{
  int a; // Declaration
  printf("%d \n", a);
}
```

Program Output

```
3148880
```

Garbage value may be different when program is executed at different times.

Formatted Input / Output Functions

- The followings are defined in the `<stdio.h>` Standard Input Output Header library file.
- **General Syntax:**

```
printf ( format-control-string , other-arguments ) ;
scanf  ( format-control-string , other-arguments ) ;
```

- printf is only used for displaying information on standard output (screen)
- scanf is only used for getting information from standard input (keyboard)

- **Format control string:**
  - describes input/output format specificiers
  - each specification begins with a percent sign (%), ends with conversion specifier
- **Other-arguments:**
  - variables correspond to each conversion specification in format-control-string

- The followings are the most commonly used format specifiers.
- Others will be studied in Chapter 9.

Format Specifier	Meaning
%d	Decimal integer number
%f	Floating number (fractional)
%s	String (array of characters)
%c	One character

Program 2

```c
#include <stdio.h>
int main()
{
  int a;  // Declaration
  a = 50; // Initialization (assignment)
  printf("%d \n", a);
}
```

Program 3

```c
#include <stdio.h>
int main()
{
  int a = 50; // Declaration and initialization in one statement
  printf("%d \n", a);
}
```

Program Output

```
50
```

Program Control

- Normally, statements in a program are executed one after the other in the order in which they're written.
- This is called sequential execution.

- Various C statements (such as if, switch, for, while) enable you to specify that the next statement to be executed may be other than the next one in sequence.
- This is called transfer of control.

Example: Adding Two Integers

```c
/* Fig. 2.5: fig02_05.c
   Addition program */
#include <stdio.h>

int main()
{
   int num1; // first number to be input by user
   int num2; // second number to be input by user
   int sum;  // variable in which sum will be stored

   printf( "Enter first integer\n" ); // prompt to user
   scanf( "%d", &num1 );           // read an integer

   printf( "Enter second integer\n" ); // prompt to user
   scanf( "%d", &num2 );            // read an integer

   sum = num1 + num2; // assign total to sum

   printf( "Sum is %d\n", sum ); // print sum
}
```

Program Output

```
Enter first integer
45
Enter second integer
72
Sum is 117
```

- The following alternative program is a way to make the coding shorter.

```
#include <stdio.h>
int main() {
  int num1, num2;
  printf( "Enter two integers\n" ); // prompt to user
  scanf( "%d %d", &num1, &num2 );   // read two integers
  printf( "Sum is %d\n", num1+num2 ); // calculate and print
}
```

2.2 Example: Adding Two Integers

- Definition (declaration) of variables
  - Variables: locations in memory where a value can be stored
  - Instead of three lines, you can define there variables in one line which are the same data type
    ```
    int num1, num2, sum;
    ```
  - `int` means the variables can hold integers (`-1`, `3`, `0`, `47`)
  - Variable names (identifiers)
    - `num1`, `num2`, `sum`
    - Identifiers: consist of letters (English only), digits (cannot begin with a digit) and underscores(_)
      - Case sensitive
- Definitions should appear before executable statements
  - If an executable statement references and undeclared variable it will produce a syntax (compiler) error

Variable Naming Examples

- **VALID Variable Names:**

  OgrenciNum, OgrNum, Ogr_Num, Ogr4,
  Sum , alfa , teta , aSquare , Pi

- **INVALID Variable Names:**

  ÖğrenciNum, Öğr Num, Ogr-Num, 4.Ogr,
  ∑ , α , θ , a^2, π

2.2 Another Simple C Program: Adding Two Integers

- `scanf( "%d", &num1 );`
  - Obtains a value from the user
    - `scanf` uses standard input (usually keyboard)
  - This `scanf` statement has two arguments
    - `%d` - indicates data should be a decimal integer
    - `&num1` - location in memory to store variable
    - **Important:** `&` is the memory address operator that must be used only for numerical variables in `scanf` statements

- When executing the program the user responds to the `scanf` statement by typing in a number, then pressing the *ENTER* (return) key

2.2 Another Simple C Program: Adding Two Integers

- = (assignment operator)
  - Assigns a value to a variable
  - Is a binary operator (has two operands)

```
sum = variable1 + variable2;
```

  - sum gets the result of adding operation variable1 + variable2
  - Target variable receiving value must be on left always

- **Important:**
  - The following gives a compiler error, becuase the left of the assignment operator (=) can not be an arithmetic expression.

```
variable1 + variable2 = sum;  // wrong!
```

2.2 Another Simple C Program: Adding Two Integers

- `printf( "Sum is %d\n", sum );`
  - Similar to `scanf`
    - `%d` means decimal integer will be printed
    - `sum` specifies the result integer will be printed

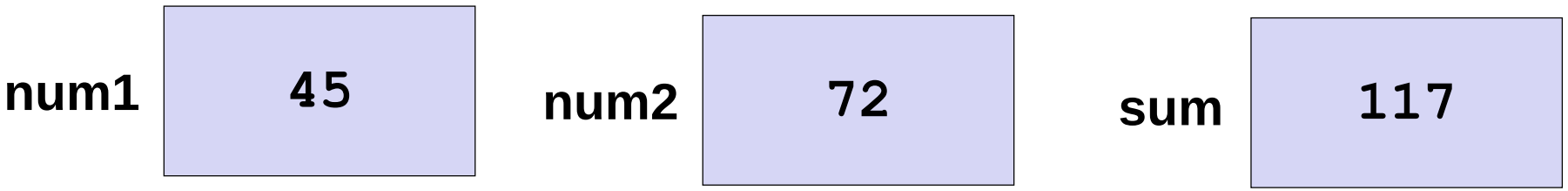
- Calculations can be performed inside `printf` statements

```
printf( "Sum is %d\n", num1 + num2 );
```

2.3 Memory Concepts

- **Variables**
  - Variable names correspond to locations in the computer's memory (RAM)
  - Every variable has a name, a type, a size and a value
  - Whenever a new value is placed into a variable (through `scanf`, for example), it replaces (and destroys) the previous value
  - Reading variables from memory does not change them

- A visual representation



Storing Data in Variables

- You can think of a variable as if it were a box inside your computer holding a data value.
- The value might be a number, character, or string of characters.

- Data is stored inside memory locations (RAM) which are defined as variables.
- Instead of remembering a specific storage location (called an address), you only have to remember the name of the variables you define.
- The variable is like a box that holds data, and the variable name is a label for that box.

- Examples:

Swapping Variables

- Swapping values simply means replacing one variable's contents with another's and vice versa.
- Suppose we assigned two variables named Sayi1 and Sayi2 with the following statements:

```
int  Sayi1 = 50 ;
int  Sayi2 = 100 ;
```

- Now we want to swap (i.e. exchange) their content values:

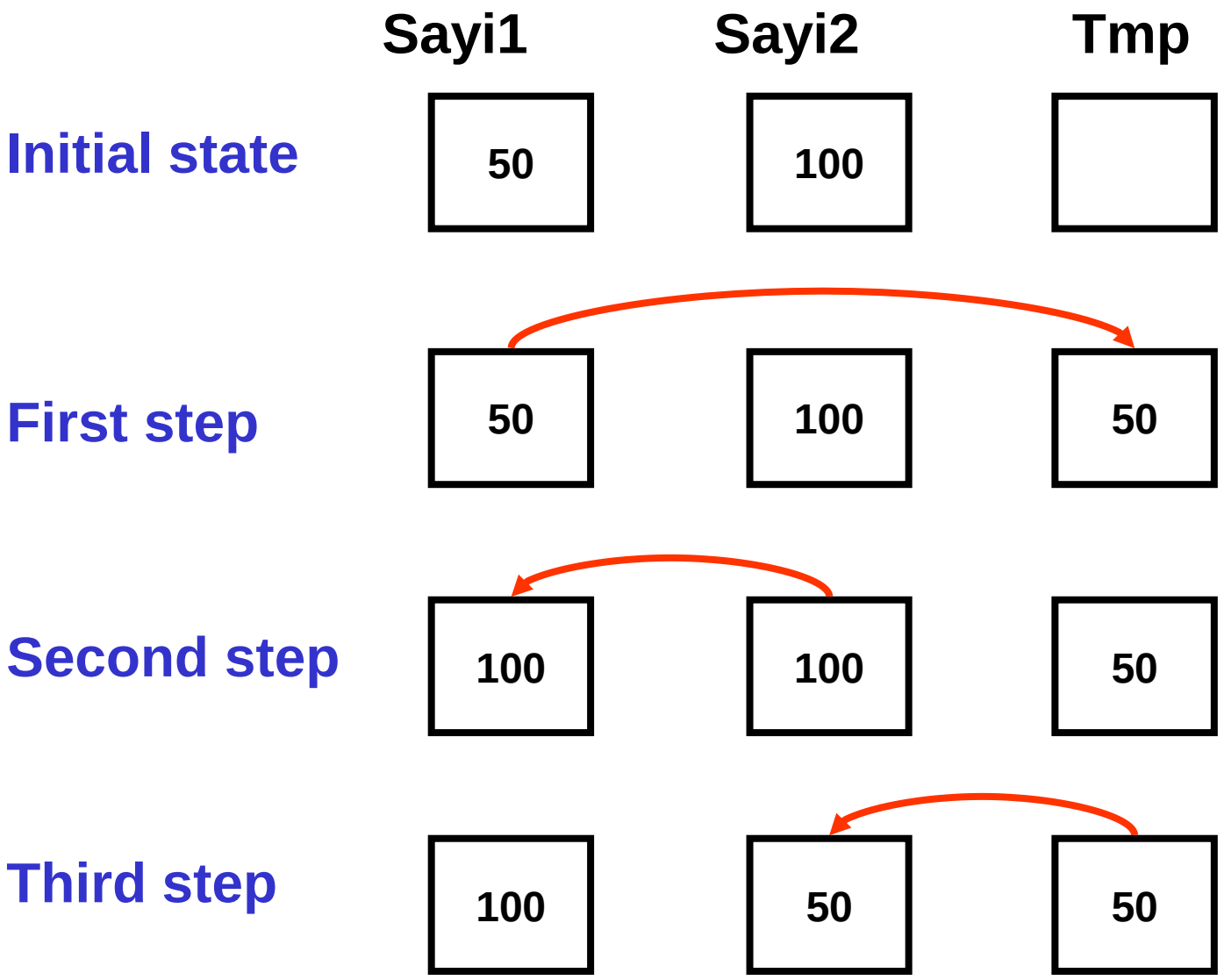
WRONG METHOD

```
Sayi1 = Sayi2;
Sayi2 = Sayi1;
```

CORRECT METHOD

```
int  Tmp;
Tmp = Sayi1;
Sayi1 = Sayi2;
Sayi2 = Tmp;
```

Swapping Variables



2.4 Arithmetic

- Arithmetic calculations
  - Use * for multiplication and / for division
  - Integer division truncates remainder
    - `7 / 5` evaluates to `1`
  - Modulus operator (%) returns the remainder
    - `7 % 5` evaluates to `2`
- Operator precedence
  - Some arithmetic operators act before others (i.e., multiplication before addition)
    - Use parenthesis when needed
  - Example: Find the average of three variables `a`, `b` and `c`
    - Do not use: `a + b + c / 3`
    - Use: `(a + b + c ) / 3.0`

2.4 Arithmetic Operators

C operation	Arithmetic operator	Algebraic expression	C expression
Addition	+	$f + 7$	f + 7
Subtraction	-	$p - c$	p – c
Multiplication	*	$b \cdot r$	b * r
Division	/	x / y	x / y
Modulus	%	$r \bmod p$	r % p

Rules of operator precedence

highest

lowest

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated left to right.
*, /, or %	Multiplication, Division, Modulus	Evaluated second. If there are several, they are evaluated left to right.
+ or −	Addition Subtraction	Evaluated last. If there are several, they are evaluated left to right.

2.4 Arithmetic

Step 1. `y = 2 * 5 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`2 * 5 is 10`

Step 2 `y = 10 * 5 + 3 * 5 + 7;` *(Leftmost multiplication)*

`10 * 5 is 50`

Step 3. `y = 50 + 3 * 5 + 7;` *(Multiplication before addition)*

`3 * 5 is 15`

Step 4. `y = 50 + 15 + 7;` *(Leftmost addition)*

`50 + 15 is 65`

Step 5. `y = 65 + 7;` *(Last addition)*

`65 + 7 is 72`

Step 6. `y = 72;` *(Last operation—place 72 in y)*

Example: Integer Division

```
#include <stdio.h>

int main()
{
     // This will display 4, not 5 !

    printf("%d\n", (5 / 2) * 2);
}
```

intermediate result is 2

Rule of Division:

- If at least one of the operands is a float, then the result will be a float.
- If both operands are integer, then the result will be an integer.

Operation	Result
5 / 2	2
5.0 / 2	2.5
5 / 2.0	2.5
5.0 / 2.0	2.5

Example: Divisions

```c
#include <stdio.h>

int main() {
   int X = 15;

   printf("%d \n", X/2);  // 7

   printf("%f \n", X/2);  // 0.000000

   printf("%f \n", X/2.0);  // 7.500000

   printf("%f \n", (float) X / 2);  // 7.500000  (float) means typecasting

   printf("%.3f \n", (float) X / 2);  // 7.500

   printf("%.1f \n", (float) X / 2);  // 7.5

   printf("%d \n", 60 / 0);  // Compiler warning of zero division!

   printf("%d \n", 60 / (X-15));  // Run-time error, Program will crash!
}
```

2.5 Decision Making: Equality and Relational Operators

- Executable statements
  - Perform actions (calculations, input/output of data)
  - Perform decisions
    - May want to print `"pass"` or `"fail"` given the value of a test grade

- `if` control statement
  - Simple version in this section, more detail later
  - If a condition is `true`, then the body of the `if` statement executed
    - `0` is `false`, non-zero is `true`
  - Control always resumes after the `if` structure

2.5 Decision Making: Equality and Relational Operators

Standard algebraic equality operator or relational operator	C equality or relational operator	Example of C condition	Meaning of C condition
Equality Operators			
=	==	x == y	x is equal to y
≠	!=	x != y	x is not equal to y
Relational Operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Part 1 of 2

```c
/* Fig. 2.13: fig02_13.c
   Using if statements, relational
   operators, and equality operators */
#include <stdio.h>

int main()
{
   int num1; // first number to be read from user
   int num2; // second number to be read from user

   printf( "Enter two integers : " );
   scanf( "%d%d", &num1, &num2 ); // read two integers

   if ( num1 == num2 ) {
      printf( "%d is equal to %d\n", num1, num2 );
   } // end if

   if ( num1 != num2 ) {
      printf( "%d is not equal to %d\n", num1, num2 );
   }
```

Part 2 of 2

```
   if ( num1 < num2 ) {
      printf( "%d is less than %d\n", num1, num2 );
   }

   if ( num1 > num2 ) {
      printf( "%d is greater than %d\n", num1, num2 );
   }

   if ( num1 <= num2 ) {
      printf( "%d is less than or equal to %d\n", num1, num2 );
   }

   if ( num1 >= num2 ) {
      printf( "%d is greater than or equal to %d\n", num1, num2 );
   }
} // end main
```

Program Output 1

```
Enter two integers : 3  7
3 is not equal to 7
3 is less than 7
3 is less than or equal to 7
```

Program Output 2

```
Enter two integers : 22  12
22 is not equal to 12
22 is greater than 12
22 is greater than or equal to 12
```

Program Output 3

```
Enter two integers : 7  7
7 is equal to 7
7 is less than or equal to 7
7 is greater than or equal to 7
```

2.5 Decision Making: Equality and Relational Operators

Operators				Associativity
*	/	%		left to right
+	-			left to right
<	<=	>	>=	left to right
==	!=			left to right
=				right to left

Fig. 2.14 Precedence and associativity of the operators discussed so far.

Basic Data Types of Variables

- char
- int
- float
- double

Modifiers for Sign and Size

- unsigned
- signed *(by default)*
- short
- long *(by default)*

Format Specifiers for printf and scanf

Data types	`printf` conversion specifications	`scanf` conversion specifications
`long double`	`%Lf`	`%Lf`
`double`	`%f`	`%lf`
`float`	`%f`	`%f`
`unsigned long int`	`%lu`	`%lu`
`long int`	`%ld`	`%ld`
`unsigned int`	`%u`	`%u`
`int`	`%d`	`%d`
`short`	`%hd`	`%hd`
`char`	`%c`	`%c`
`char (string)`	`%s`	`%s`

Data Type Ranges (Signed)

Keyword	Size in Bytes	Variable Type	Range
char	1	Character (or string)	-128 to 127
int	4	Integer	-2,147,483,648 to 2,147,483,647
long			
long int			
short	2	Short integer	–32,768 to 32,767
short int			

Data Type Ranges (Unsigned)

Keyword	Size in Bytes	Variable Type	Range
unsigned char	1	Unsigned character	0 to 255
unsigned int	4	Unsigned integer	0 to 4,294,967,295
unsigned long			
unsigned short	2	Unsigned short integer	0 to 65,535

Data Type Ranges (Signed Fractional)

Keyword	Size in Bytes	Variable Type	Range
float	4	Single-precision floating-point (7 fraction digits)	$-3.4 * 10^{-38}$ to $3.4 * 10^{38}$
double	8	Double-precision floating-point (15 fraction digits)	$-1.7 * 10^{-308}$ to $1.7 * 10^{308}$

Exponent Notation

```
double  x = 4.3E6;
long long y = 4.3E6;
```

} $4.3 * 10^6$

```
unsigned short int  z = 70000;
```

} compiler warning due to overflow!

Example: Range Overflow

```
65535
+    1
------
     0
```

```c
#include <stdio.h>

int main()
{
   unsigned short int X, Y;
   // Length of these are 2 bytes (16-bit) each

   X = 65535;
   // Maximum possible value for unsigned short integer numbers

   Y = X + 1;
   // Overflow is expected here (Y will be 0, instead of 65536)

   printf("SONUC = %d \n", Y);
   // It will display zero !!
}
```

Binary Represenation (16-Bit Memory)