Program and Data

Fundamentals of Programming

Lecture 1B

Contents

- Program components
- Syntax
- Literal data
- Data types
- Variables
- Printing text and values

Program components

- C program has the basic structure with:
  - Header file
  - main() function
- The header file contains library functions that can be used in the program
- There is only one main() function
- Functions are indicated by the function name and brackets ()

```
#include <stdio.h>

int main()
{
  printf("Hello");
  return 0;
}
```

Comments

- Comments are included in the program to provide information on certain parts
- This helps to understand the program during test and debugging

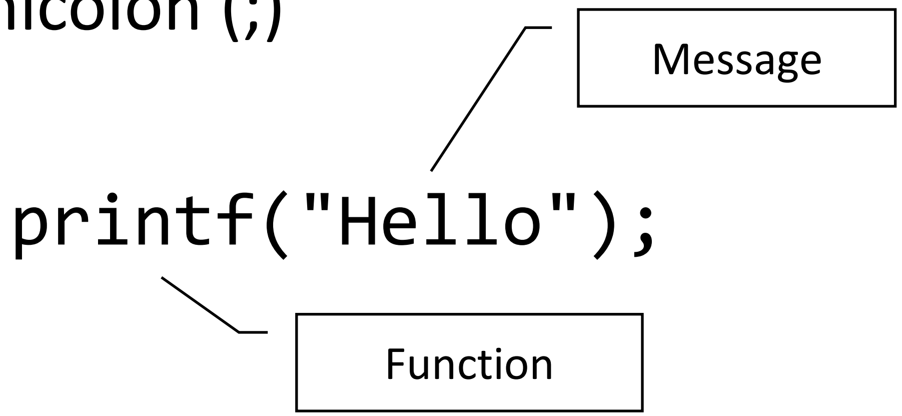
```
// Single line comment

/* Block comment */

/* Block
   comment */
```

Statement

- A statement is an expression to be executed (a line of code)
- A statement could be used to:
  - Assign a value to a variable
  - Call a function
  - Execute other statements
- Always ends with a semicolon (;)

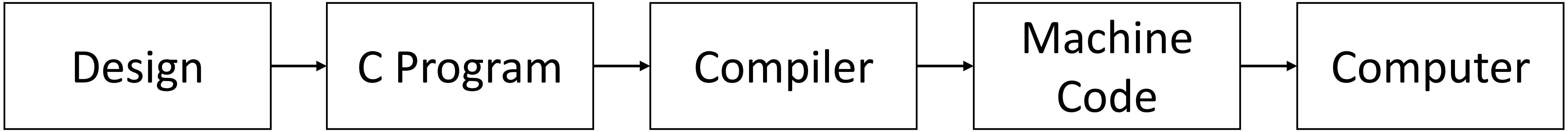


Message

```
printf("Hello");
```

Function

Process



Design

C Program

Compiler

Machine Code

Computer

Tools required

- Computer (Windows, macOS, Linux)
- Integrated Development Environment (IDE)
  - Text Editor
  - C compiler

- For a quick start, you can use online GDB
  - https://www.onlinegdb.com/

Reserved keywords

Reserved keywords

- Syntax refers to how the C language interprets the logical instructions

- Reserved keywords have specific meanings to the compiler

```
#include <stdio.h>

int main()
{
  printf("Hello");
  return 0;
}
```

Character set

- Letters
  - A, b, c, D….Z, upper case and lower case
- Digits
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- Special characters
  - Characters used in C syntax, %, &, etc.
- White spaces
  - Blank space, horizontal tab(\t), new line(\n), carriage return(\r), and form feed(\f)

Character	Description	Character	Description
,	Comma	&	Ampersand
.	Period	^	Caret
;	Semicolon	*	Asterisk
:	Colon	-	Minus sign
?	Question mark	+	Plus sign
'	Apostrophe	<	Opening angle
"	Quotation mark	>	Closing angle
!	Exclamation mark	(	Left parenthesis
|	Vertical bar	)	Right parenthesis
/	Slash	[	Left bracket
\	Backslash	]	Right bracket
~	Tilde	{	Left brace
-	Underscore	}	Right brace
$	Dollar sign	#	Number sign
%	Percentage sign		

Reserved keywords

auto

break

case

char

const

continue

default

do

double

else

enum

extern

float

for

goto

if

inline

int

long

register

restrict

return

short

signed

sizeof

static

struct

switch

typedef

union

unsigned

void

volatile

while

Literal data

Types of literal data

- Integer: 123, 67,…..
- Real number: floating-point number, 23.1, 3.14, …..
- Single character: ‘a’, ‘t’, ….
- String (of characters): “Good morning”

Integer

- An integer is a whole number
- It can be zero, positive or negative
- It can be in the following format:
  - Decimal (base-10): `0–9`
  - Octal (base-8): `0–7`
  - Hexadecimal (base-16): `0–9, A, B, C, D, E, F`

Integer prefix

- Decimal
  - No prefix is required
  - Example: `+3, 45, -1, 320`
- Octal
  - Prefix O
  - Example: `O12, O523`
- Hexadecimal
  - Prefix `0x` or `0X`
  - Example: `0x23, 0X8F, 0x1AB6`

Real number

- A real number contains a fractional part after the decimal point
  - Example: 1.23, −48.32, 0.0036

- In a C program, it is represented as a floating-point number

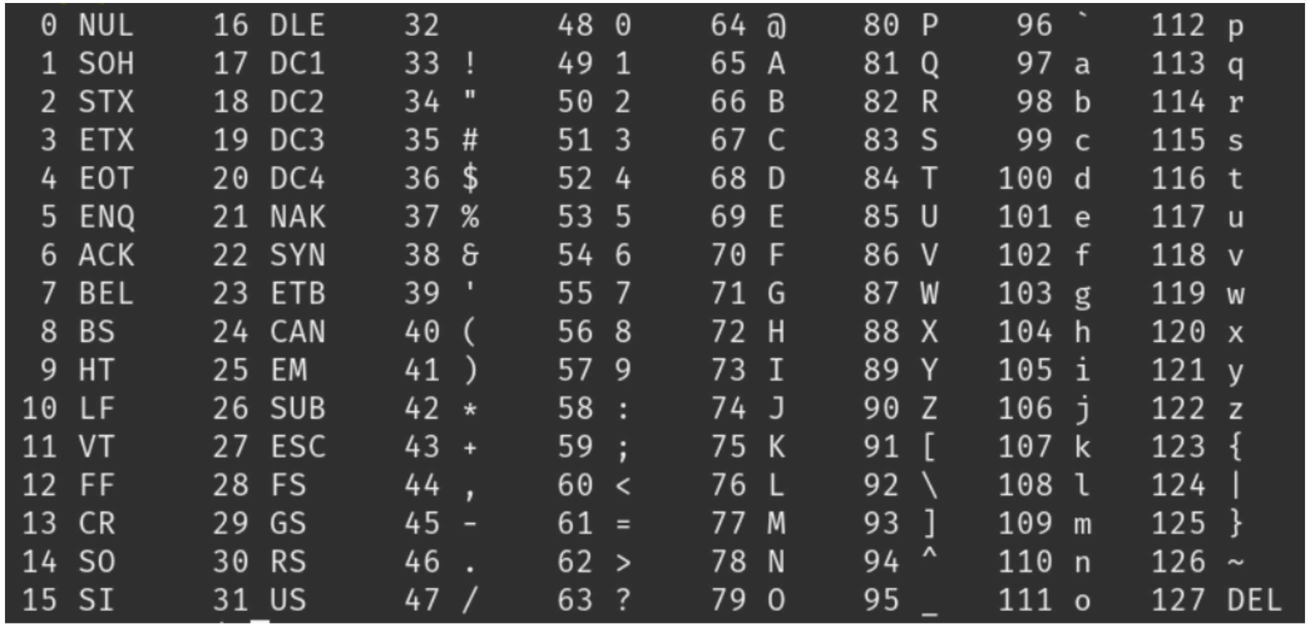
Scientific notation

- A large number may be represented by scientific notation
- Example:

```
13.2E05 = 1320000
+7E3 = 7000
-2.0394e-3 = -0.0020394
2.57e-4 = 0.000257
3.2e+6 = 3200000
```

Single character

- A single character refers to the ASCII table
- Each character must be enclosed in a pair of single quotes, ‘A’, ‘c’
- ASCII values can be in decimal, octal, hex
- Example: 'A' = 65



```
 0 NUL   16 DLE   32      48 0   64 @   80 P    96 `   112 p
 1 SOH   17 DC1   33 !    49 1   65 A   81 Q    97 a   113 q
 2 STX   18 DC2   34 "    50 2   66 B   82 R    98 b   114 r
 3 ETX   19 DC3   35 #    51 3   67 C   83 S    99 c   115 s
 4 EOT   20 DC4   36 $    52 4   68 D   84 T   100 d   116 t
 5 ENQ   21 NAK   37 %    53 5   69 E   85 U   101 e   117 u
 6 ACK   22 SYN   38 &    54 6   70 F   86 V   102 f   118 v
 7 BEL   23 ETB   39 '    55 7   71 G   87 W   103 g   119 w
 8 BS    24 CAN   40 (    56 8   72 H   88 X   104 h   120 x
 9 HT    25 EM    41 )    57 9   73 I   89 Y   105 i   121 y
10 LF    26 SUB   42 *    58 :   74 J   90 Z   106 j   122 z
11 VT    27 ESC   43 +    59 ;   75 K   91 [   107 k   123 {
12 FF    28 FS    44 ,    60 <   76 L   92 \   108 l   124 |
13 CR    29 GS    45 -    61 =   77 M   93 ]   109 m   125 }
14 SO    30 RS    46 .    62 >   78 N   94 ^   110 n   126 ~
15 SI    31 US    47 /    63 ?   79 O   95 _   111 o   127 DEL
```

String

- A group of characters from the ASCII table
- Each string must be enclosed in a pair of double quotes
- Example:

"Hello"

"Good morning!"

"What is the time?"

Data types

Data types

- Built-in data types are used to store the appropriate data
- int – Integer
- float – Floating point
- double – Double precision floating point
- char – Character

Range of values

Type	Storage size	Bit size	Value range
char	1 byte	8	−128 to 127
unsigned char	1 byte	8	0 to 255
int	4 bytes	32	−2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	32	0 to 4,294,967,295
short int	2 bytes	16	−32,768 to 32,767
unsigned short int	2 bytes	16	0 to 65,535
long int	4 bytes	32	−2,147,483,648 to 2,147,483,647
unsigned long int	4 bytes	32	0 to 4,294,967,295
float	4 bytes	32	1.2e−38 to 3.4e+38
double	8 bytes	64	2.3e−308 to 1.7e+308

Variables

Identifier

- An identifier is the name of program elements such as variables, arrays, structures, unions, labels, etc.
- The identifier must contain:
  - Only letters (uppercase or lowercase), digits, and underscore
  - A maximum of 31 characters
  - Start with a letter or underscore
  - No reserved keywords from the C syntax
  - No whitespaces

```
int class_number;
float length;
float AreaTriangle;
double distance_from_city;
```

Variables

- Variables are used to store data in a program
- Declaring a variable starts with the data type then the identifier

```
<data_type> <identifier>;
```

```
int class_number;
float length;
float AreaTriangle;
double distance_from_city;
```

Declaring multiple variables

```
<data_type> <identifier1>, <identifier2>,..., <identifierN>;
```

```
int class_number, student_number;
```

Declaring a variable and its initial value

```
<data_type> <identifier> = <value>;
```

```
<data_type> <identifier1>=<value>, <identifier2>=<value>;
```

```
int class_number = 5;
```

```
int class_number = 5, student_number = 200;
```

Integer declaration

```
int x;
int a, b, c;
int count = 0;
int min = -100, max = 100;
```

Float declaration

```
float length;
float temperature;
float gravity = -9.80665;
double AreaTriangle, AreaCircle;
double distance_from_city;
double epsilon = 8.854187e-12;
```

Character declaration

```
char gender;
char exam_grades;
char Level = 'A';
```

User-defined data types

- The reserved keyword typedef is used to create an alias of the original data type
- Custom identifier for data types are used to make the variables in a program more meaningful

```
typedef <data_type> <identifier>;
```

Typedef example

- The volume of physical object is declared as a float data type
- Volume as a typedef variable is used to declare several other variables

```
// volume is equivalent to float here.
typedef float volume;
volume sphere, cuboid, pyramid;
```

Enumerated data types

- Identifiers can be grouped together as an enumerated list

```
enum <name> {<identifier1>,<identifier2>,…,<identifierN>};
```

```
enum week {sun, mon, tue, wed, thu, fri, sat};
```

Enum example 1

```
enum week {sun, mon, tue, wed, thu, fri, sat};
```

- The days of the week is readable by humans
- However, the days must have numerical values so that it is readable by the program
- By default, the first identifier is 0, the identifiers sun = 0, mon = 1, …, fri = 5, sat = 6
- First identifier can be initialized with a chosen value

```
enum week {sun=20, mon, tue, wed, thu, fri, sat};
```

Const

- Const is a qualifier keyword for creating constants
- Constants are used for values that are not expected to change
- Constants can only be initialised once during declaration
- Default value of uninitialised constants depend on residual data

```
const <data_type> <identifier> = <value>;
const <data_type> <identifier>; //residual data, can
be random in memory
```

Const example

```
const int x;
const int y = 2;
```

- Constant x will have default value, not recommend this
- Constant y will be initialised to 2
- Future modification to values in x and y is not possible
- Example:

Escape sequences

Escape sequences

- Escape sequence characters allow the program to use invisible characters found on the keyboard
  - Press ENTER = carriage return and new line
  - Press Backspace = backspace
  - Press ESC = escape
  - Press TAB = horizontal tab

Use escape sequence characters in program

Escape sequence	Action
\'	Apostrophe or single quotation mark
\"	Double quotation mark
\?	Question mark
\\	Backslash
\a	Alert (bell)
\b	Backspace
\e	Escape
\f	Formfeed page break
\n	New line
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

Format specifiers

Formatting the output

- Printing values in variables requires the correct format specifier
- Used in printf() and scanf() functions

Format specifier	Description	Format specifier	Description
`%d, %i`	Decimal	`%o`	Octal
`%f`	Float	`%x`	Hexadecimal
`%c`	Character	`%e`	Exponent
`%s`	String	`%g`	Exponent (short ver.)
`%u`	Unsigned int	`%p`	Pointer address
`%hd, %hi`	Short int	`%lu`	Long int

Precision of format specifier

- Values to be printed to the terminal screen can be limited by precision

Format specifier	Description
%d	Print decimal integer
%8d	Print as decimal integer, minimum 8 characters wide
%f	Print floating point number
%8f	Print floating point number, minimum 8 characters wide
%.2f	Print floating point number and 2 digits after decimal point
%8.2f	Print floating point number, minimum 8 characters wide and 2 digits after decimal point

Format specifier example

```
int x = 7;
int y = 12;
float z = 5.5079;
printf("x = %d\n", x);
printf("x + y = %4d\n", x+y);
printf("x + y + z = %f\n", x+y+z);
printf("x + y + z = %6.2f\n", x+y+z);
```

(demo this in online GDB)

Printing text and values

Print text and variables

- printf() function is used to output text or values to the screen
- Format output using control string

```
printf("<message>");
printf("<control_string>",<arg1>,<arg2>,...,<argN>);
```

```
printf("Hello!")
printf("%d", x);  //x is an integer
printf("%8d", x); //x is an integer
```

Program to output text and values

```
#include <stdio.h>
int x = 4;
int main()
{
   int y = 7;
   printf("x = %d\n", x);
   printf("y = %d\n", y);
   return 0;
}
```

Library header file

Global variable

Block of code

Local variable

Program to output arithmetic result

```
#include <stdio.h>
float PI = 3.14159;
int main()
{
  float x = 2.5 + PI;
  printf("result = %f\n", x);
  return 0;
}
```

Program to print escape characters

```
#include <stdio.h>
int main()
{
  printf("\n"); // new line
  printf("\a"); // bell
  printf("\?"); // question mark
  return 0;
}
```

Reading text and values

Read text and values

- scanf() function is used to input text or values into variables
- Format input using control string
- Ampersand (&) indicates memory address of the variable

```
scanf("<control_string>", &<arg1>, ..., &<argN>);
```

```
scanf("%d", &x);
```

Program to read a value

```
#include <stdio.h>
int main()
{
  int x;
  printf("Enter x value: ");
  scanf("%d", &x);
  printf("x = %d\n", x);
  return 0;
}
```

Program to read two values

```c
#include <stdio.h>
int main()
{
  int x, y;
  printf("Enter x value: ");
  scanf("%d", &x);
  printf("Enter y value: ");
  scanf("%d", &y);
  printf("x + y = %d\n", x+y);
  return 0;
}
```

Summary

- The C program consists of declaration of library header files, functions, characters, and variables
- Use of correct data type to store the literal data
- Typedef and enum allows existing data types to be customisable
- The output and the input must be formatted, in order to get proper results

Further readings

- C: How to Program, 8th Edition, Paul Deitel & Harvey Deitel
  - Printing text and variables: pp. 72−75
  - Variables: pp. 77−84
- The C Programming Language, 2nd Edition, Kernighan & Ritchie
  - Printing text and variables: pp. 9−15