Programming in C language

Microprocessor Techniques and Embedded Systems

Tomas Fryza

Sept 2018

Contents

Contents

Programming language

Definition

Programming language is the way how any algorithm is rewritten for a computer. The algorithm in selected language is called a program

- Division of programming languages:
  - low-level languages: knowledge of HW and instruction set
  - high-level languages: higher level of code abstraction; compiled (C, ...) of interpreted (PHP, Python, ...)

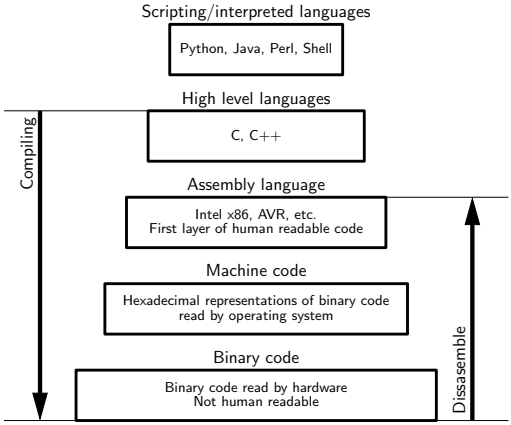

Figure: Programming layers

Basic rules for programming

- Code must be understandable not only to author!
- **Coding rules**: strict rules of language (syntax, commands, ...)
- **Coding guidelines**: rules for better code understanding within a team (company, ...)
- Some Coding guidelines:
  - Use "comment header" in every source files
  - Horizontal indentation (better spaces then tabs)
  - Limit number of characters per line to approx. 80
  - Empty lines could be use for better visibility
  - Use comprehensible identifiers for variables, functions, ...
  - All identifiers only in English. Use letters A, B, ..., Z, a, ..., z, digits 0, ..., 9 and underscore "_" only

```c
/*************************************************
* main00_sol.c
*
* Tomas Fryza, Brno University of Technology
* Date/Time updated: Fri Oct 21 16:01:55 2016
*
*    Target MCU : ATmega16
*    Description: Binary counter with delay
*/

#include <avr/io.h>      // definition file
#define F_CPU 1000000UL  // clock frequency
#include <util/delay.h>  // delay library

/*************************************************
* Main function
*/
int main(void)
{
    /** ------------------------------------------*
    * setup I/O port */
    DDRB  = 0xff;        // set output direction
    PORTB = 255;         // turn off all LEDs

    /** ------------------------------------------*
    * forever loop */
    while (1){
        _delay_ms(50);   // wait for 50 ms
        PORTB = PORTB-1;// change binary counter
    }

    return 0;
}
```

C code example

```
/**
*********************************************************************
* @file main03_sol.c
*
* @brief  Control of TWI bus
* @author Tomas Fryza
* @date   Thu Nov 17 19:15:55 CET 2016
*
* Counter on TWI bus expander controlled by 16-bit timer interrupt
*/

#include <avr/io.h>              // definition file for MCU ATmega16
#include "twi_h.h"               // TWI library

char temp = 64;                  // counter value

int main(void)
{
    twi_init();                  // initialization of TWI bus
                                 // prescaler = 64 (260 ms)
    TCCR1B = TCCR1B | ((0<<CS12)+(1<<CS11)+(1<<CS10));
    TIMSK  = TIMSK | (1<<TOIE1);// enable overflow interruption
    sei();                       // enable all interrupts

    while (1);
    return 0;
}
```

- block comment of source file
- compiler directives
- declaration of variable `temp`
- main function of application: `main()`
- calling of user-defined function: `twi_init()`
- C code statements

C language

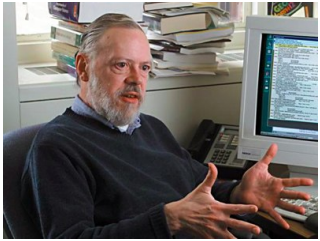

Figure: Dennis MacAlistair Ritchie (⋆9. 9. 1941, †12. 10. 2011))

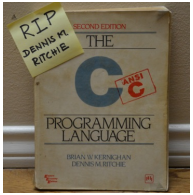

Figure: D.M.Ritchie, Brian Kernighan. *The C Programming Language*. 1978

- First edition of *The C Programming Language* by Brian Kernighan and Dennis Ritchie (1978, called K&R C) defined the C language, but not the C library

C89 ANSI (American National Standards Institute) developed a new standard in 1989 (called ANSI C). This new standard defines both the language and a standard C library

C90 The International Organization for Standardization adopted a C standard (ISO C) in 1990. ISO C and ANSI C are essentially the same standard. The final version of the ANSI/ISO standard is often referred to as C89 or C90

C99 In 1994, work began on revising the standard, an effort that resulted in the C99 standard

- Actual standard: ISO/IEC 9899:2018, approx. 500 pages, price 200 CHF, see `https://www.iso.org/standard/74528.html`

Process of compilation

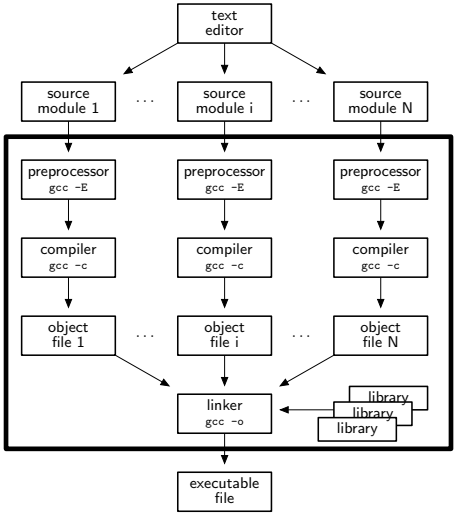

Figure: Compilation process of apliaction with several modules (gcc compiler is used)

- Application in C contains several modules. Each module has two text files: source file `*.c` and header file `*.h`
- Typically, header files include other header files (e.g. libraries), definitions of user functions and constants
- Simple applications with one (short) source file `*.c` do not have any header file and definitions are directly implemented in `*.c`

Basic principles in C

- All commands end by semicolon **;**
- Bodies of functions, conditions, cycles, etc. are grouped inside the braces {·}
- Strings are enclosed by quotation marks "·"
- Comments are introduce by double slash //, or are enclosed by /* ... */
- Using of reserved words is strictly observed: **for**, **return**, **switch**, **case**, **if**, **else**, **char**, **int**, **float**, **unsigned**, **void**, ...
- Function prototype defines type of return value, function name, and number and types of all input parameters

```
int imax(float*, int);
```

- User function declaration:

```
return_value function_name(param_0, param_1, ...)
{
    ...         // function body
}
```

Structure of C code

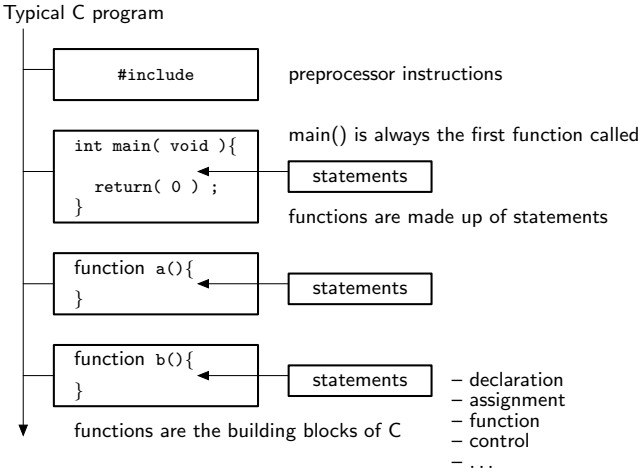

Figure: Anatomy of a C code

Tips to make code readable

- Choose meaningful variable names and use comments. Note that these two techniques complement each other
- Using blank lines to separate one conceptual section of a function from another. (C does not require the blank line, but it enhances readability)
- Use one line per statement
- C has a *free-form* format

```
int main( void ) { int rank; rank
=
1
;printf(
    "Microprocessors are no. #%d for me\n",
rank);return( 0 );}
```

```
/* main application */
int main(void)
{
    int rank = 1;                   // temporal variable
                                    // display text
    printf("Microprocessors are no. #%d for me\n", rank);

    return(0);                      // return from function
}
```

Variables

- All variables are identified by its name and type and must be declared before using:

```
char temp;              // signed value, 8 bits
unsigned int ii,jj;     // unsigned value, 16 bits
float f = 3.14;         // real value, 32 bits
```

- According to accessibility, each variable could be:

Local: Allocated in stack or heap memory part while the function is executed.
Global: Allocated during compiling process, accessible for all functions.

Type	Size [b]	Range
`char`	8	-128 to $+127$
`unsigned char`	8	0 to 255
`signed char`	8	-128 to $+127$
`int`	16	$-32\,768$ to $+32\,767$
`unsigned int`	16	0 to $65\,535$
`float`	32	$\pm 1,175 \cdot 10^{-38}$ to $\pm 3,402 \cdot 10^{38}$

Amount od allocated memory for selected data types

- List of source file `main.c`:

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    printf("%d ", sizeof(char));
    printf("char\n");

    printf("%d ", sizeof(short int));
    printf("short int\n");

    printf("%d ", sizeof(int));
    printf("int\n");

    printf("%d ", sizeof(long int));
    printf("long int\n");

    printf("%d ", sizeof(float));
    printf("float\n");

    /* end of main function */
    return 0;
}
```

- At the position of %d standard function `printf` displays integer value indicated after the string
- Formating character \n inserts a new line
- Output of the application:

```
1 char
2 short int
4 int
8 long int
4 float
```

Variables (cont.)

- For all integer types (`char`, `short int`, `int` and `long`) the modificators `signed` and `unsigned` could be used

`unsigned`: Variable with positive values only

`signed`: Variable with both positive and negative values; default value for all integer data types in C

```c
#include <stdio.h>

int main()
{
    char a = 200;
    unsigned char b = 200;
    signed short int c = -33000;

    printf("char a = 200: ");
    printf("%d\n", a);

    printf("unsigned char b = 200: ");
    printf("%d\n", b);

    printf("signed short int c = -33000: ");
    printf("%d\n", c);

    return 0;
}
```

- `char` is 8-bit data type, therefore `signed char` represents range −128 to 127 and data type `unsigned char` represents range 0 to 255
- There is a warning during the compilation but application is executable:

```
char a = 200: -56
unsigned char b = 200: 200
signed short int c = -33000: 32536
```

Integer data types from library `stdint.h`

```c
/* standard input/output library */
#include <stdio.h>
/* standard integer library */
#include <stdint.h>

/* main function */
int main()
{
    int8_t  a = 200;
    char    b = 200;
    uint8_t c = 200;
    unsigned char d = 200;

    printf("int8_t is equivalent of char\n");
    printf("%d, %d\n\n", a,b);

    printf("uint8_t is equivalent of unsigned char\n");
    printf("%d, %d\n", c,d);

    /* end of main function */
    return 0;
}
```

- Standard C99 defines other types within `stdint.h` library:

```
int8_t int16_t int32_t uint8_t
uint16_t uint32_t
```

- Output of application:

```
int8_t is equivalent of char
-56, -56

uint8_t is equivalent of unsigned char
200, 200
```

Arithmetic and Binary Operations

Table: Arithmetic operations

Operation	Operand
Multiplication	*
Division	/
Modulo division	%
Addition	+
Subtraction	-
Incrementation	++
Decrementation	--

- Simplified notations:

```
a += 3;      // a = a + 3;
a++;         // a = a + 1;
b -= 2;      // b = b - 2;
c *= 5;      // c = c * 5;
d /= a;      // d = d / a;
```

Table: Binary operations

Operation	Operand
One's complement	~
Left shift	<<
Right shift	>>
Logical AND	&
Logical OR	|
Logical EX-OR	^

- Simplified notations:

```
a |= 3;      // a = a | 3;
b &= 2;      // b = b & 2;
c ^= 5;      // c = c ^ 5;
d <<= 2;     // d = d<<2;
```

Condition

- Condition: `if`

```
if( condition ) {
  ;
}
else {
  ;
}
```

- Condition `switch`

```
switch( variable ) {
  case value0:
    ;
  case value1:
    ;
  default:
    ;
}
```

Table: Condition operators

Operation	Operand
Equal to	==
Not equal to	!=
Less than	<
Less than or equal	<=
Greater than	>
Greater than or equal	>=

Example of condition: `if`

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    int a, b;

    printf("Enter two integer numbers:\n");
    scanf("%d %d", &a, &b);

    if (a == b) {
        printf("Numbers are equal: %d\n", a);
    }
    else if (a < b) {
        printf("%d < %d\n", a, b);
        printf("a = %d\nb = %d\n", a, b);
    }
    else {
        printf("%d > %d\n", a, b);
        printf("a = %d\nb = %d\n", a, b);
    }

    /* end of main function */
    return 0;
}
```

- Output of application:

```
Enter two integer numbers:
3
7
3 < 7
a = 3
b = 7
```

Loops

- Loop: `for`

```
for( par0; par1; par2 ) {
  ;
}
```

- Loop: `while`

```
while( condition ) {
  ;
}
```

- Forever loop: `for`

```
for( ;; ) {
  ;
}
```

- Forever loop: `while`

(a)
```
while( 1 ) {
  ;
}
```

(b)
```
while( 1 ) ;
```

Example of loop: `for`

- Source code:

```c
/* standard input/output library */
#include <stdio.h>

/* main function */
int main()
{
    int i;

    for (i = 10; i < 20; i = i + 2 ){
        printf("index value: %d\n", i);
    }

    /* end of main function */
    return 0;
}
```

- Output of application:

```
index value: 10
index value: 12
index value: 14
index value: 16
index value: 18
```