C Programming under Linux

P2T Course, Martinmas 2003–4
C Lecture 2

Dr Ralf Kaiser

Room 514, Department of Physics and Astronomy

University of Glasgow

r.kaiser@physics.gla.ac.uk

Summary

- Computer Memory Organisation
- Number Systems
- Variables
- Simple Operators
- Printing to the Screen

`http://www.physics.gla.ac.uk/~kaiser/`

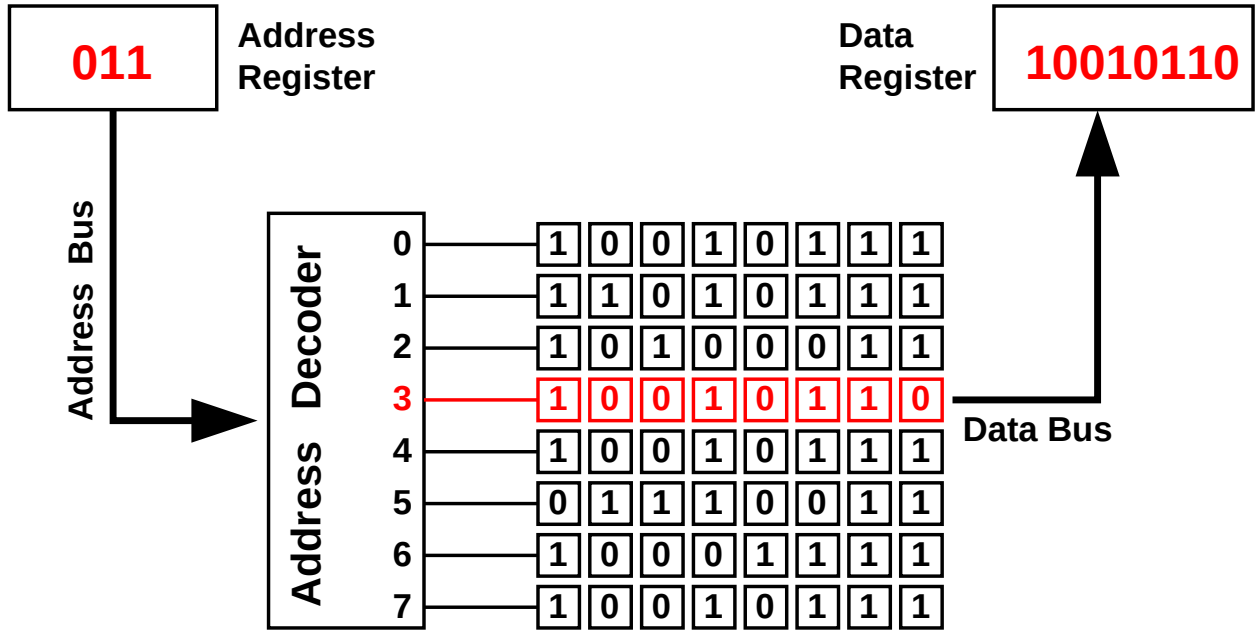
Computer Memory

- Computer memory is classified as either ROM (read only memory) or RAM (random access memory).
- Static RAMs (SRAM) use flip-flops as storage elements and can therefore store data indefinitely as long as DC power is applied.
- Dynamic RAMs (DRAM) use capacitors as storage elements and cannot retain data very long without the capacitors being recharged.
- Data can be read much faster from SRAMs than from DRAMs. DRAMs can store much more data for a given physical size and cost of the memory, because the DRAM cell is much simpler. Therefore you may find that your PC uses DRAM as main memory (e.g. EDO DRAM, extended data out DRAM), but the cache better be SRAM or your PC will be slower than it could be.

Computer Memory cont.

- One bit is the unit of information, i.e. one 1 or 0. Eight bits are one byte.
- A complete unit of information is called a word and generally consists of one or several bytes. As a simplified general rule we can say that memories store data in bytes.
- The memory can be imagined as an array of cells (flip-flops or capacitors) that has 8 columns and a large number of rows. In this picture a 64 MByte memory has 8 columns and 64 million rows (actually , more like 67 million rows, as 1 k is 1024 in memory).
- The location of data in a memory array is called its address.

Memory Organisation



- Data units go in and out of the memory on a set of lines called the data bus.
- For any read or write operation an address is selected by placing the corresponding binary code on a set of lines called the address bus.

Number Systems

- Our regular, every day number system is the decimal system. It's probably so successful because of the 10 fingers we have; in fact the word *digit* comes from the Latin word for finger.
- In the decimal system, the position of the digit indicates it's value in powers of 10:
  $2003 = 2 \cdot 10^3 + 0 \cdot 10^2 + 0 \cdot 10^1 + 3 \cdot 10^0$
- In other number systems the position of the digit indicates it's value in powers of another value: powers of 2 for binary, powers of 8 for octal and powers of 16 for hexadecimal numbers.
- Octal number only need the symbols 0..7, hexadecimal numbers use A..F for the numbers 10..15, e.g. 2003 (decimal) becomes 7D3 (hexadecimal).
- Binary, octal and hexadecimal are the natural number systems for computers.

Variable Types

In C data are stored in variables and there are three basic types of variables:

- `int`
  integer variables, whose values are integer numbers such as 4, -128, 147238
- `float`
  floating point numbers, corresponding to non-integer numbers with a decimal point. 5.0 is a floating point number, while 5 is an integer.
- `char`
  character variables with values ’a’ to ’z’, ’A’ to ’Z’, ’0’ to ’9’ plus punctuation marks, parentheses etc.

Variable Types - `int`

- size typically reflects the natural size of integers on the host machine. The C standard does not define the size of numbers. Under Linux (or another Unix) integers are typically 32 bits (4 bytes), providing a range from 2147483647 ($2^{31} - 1$) to -2147483648.
- The standard header file `limits.h` defines constants for the various numerical limits.

```
/* Minimum and maximum values a ‘signed int’ can hold.  */
#  define INT_MIN       (-INT_MAX - 1)
#  define INT_MAX       2147483647
```

- for comparison: in Turbo C++ integers are only 16 bit, with a range from 32767 to -32768. The example of 147238 from the last slide would not be possible.

Variable Types - `float` and `double`

- usually `float` variables use 4 bytes, corresponding to a value range of $3.4 \cdot 10^{-38}$ to $3.4 \cdot 10^{+38}$ with a corresponding range of negative numbers and approximately 7 significant digits.
- double variables use 8 bytes, corresponding to a value range of $1.7 \cdot 10^{-308}$ to $1.7 \cdot 10^{+308}$ with a corresponding range of negative numbers and approximately 15 significant digits.
- floating point numbers can be given in exponential form, e.g. 1.2e34 is $1.2 \cdot 10^{34}$
- While it is possible to write .5 instead of 0.5 and 12. instead of 12.0, this should be avoided.
- computers do integers better than floating point numbers, and in floating point operations rounding errors can accumulate.

Variable Types - `char`

- single characters are represented by an 8 bit number, this length is also defined in `limits.h`:

```
/* Number of bits in a 'char'.  */
#  define CHAR_BIT      8
```

- characters are enclosed in single quotes

```
char char1;    /* first character  */
char char2;    /* second character */
char char3='z' /* third character, directly initialised */

char2 = '9';   /* assignment statement for second character */
```

- only 7 bits are actively used to encode the character in the ASCII code (American Standard Code for Information Interchange), bit 7 can be used as a parity bit.
- characters can also be specified by `\nnn` where nnn is their octal code according to ASCII (e.g. `\100` is @)

Variable Types - Qualifiers

- The variable type can be combined with the qualifiers `short` or `long` to limit or extend their range and with the qualifiers `signed` or `unsigned`.
- where `int` is 32 bit, `short int` is only 16 bit, `long int` is 32 bit, but `long long int` is 64 bit
- `long`, `long long` and `unsigned` numbers are marked with `U` and `L, LL` at the end. The largest integer number in `gcc` on a Linux machine is an `unsigned long long int`:

```
/* Maximum value an ‘unsigned long long int’ can hold.  (Minimum is 0.)  */
#   define ULLONG_MAX   18446744073709551615ULL
```

- `long double` has a maximum value of `1.189731495357231765 02e+4932L` - as defined in `float.h`.

Variable Names

- names in C start with a letter or an underscore (_), followed by any number of letters, numbers or underscores
- for an internal name at least the first 31 characters are significant
- special signs and spaces are not allowed; C commands are reserved words and can't be used as variables names
- uppercase is different from lowercase, so `max`, `Max` and `MAX` specify three different variables
- names starting with underscores are conventionally used only for internal and systems variables
- lowercase variable names are typical for C, so are lowercase_and_underscore, but some also use thisStyle.

Variable Names - Examples

- examples for valid variable names and declarations are

```
int number_of_students = 47;  /* number of students in this class */
float pi = 3.1415927;         /* pi to 7 decimal places           */
int dataRecoil;               /* Recoil detector data               */
```

- examples for invalid variable names are

```
3rd_entry    /* starts with a number */
all$done     /* contains a '$'       */
int          /* reserved word        */
home phone   /* contains a space     */
```

- Use variable names that describe the contents. I once knew a programmer who had used the names of rivers in Armenia as variables in a high voltage control program. Beautiful, but not very practical.
- Don't forget to comment the variables, even if the names are chosen well.

Simple Operators

There are 5 simple arithmetic operators in C:

Operator	Meaning
*	multiply
/	divide
+	add
-	subtract
%	modulus (return remainder after integer division)

- multiply (*), divide (\) and modulus (%) have precedence over add (+) and subtract (-)
- parentheses () may be used to group terms
- these operators are also refered to as binary operators, because they have two operands, e.g. a + b

Simple Operators - Examples

```
int  term1;                    /*  first   term                          */
int  term2;                    /* second term                           */
int  sum;                      /* sum of  first   and second term       */
int  difference ;              /* difference  of   first   and second term */
int  modulo;                   /* term1 modulus term2                    */
int  product;                  /* term1 * term2                         */
int   ratio ;                  /* term1 / term2                         */

int  main()
{
term1  = 1 + 2 * 4;            /* yields 2*4=8   8+1=9                  */
term2  = (1 + 2) * 4;          /* yields 1+2=3   3*4=12                 */
sum = term1 + term2;           /* yields 9+12=21                        */
difference  = term1 - term2    /* yields 9-12=-3                        */
modulo = term1 % term2         /* yields 9/12=0, remainder is 9         */
product = term1 * term2        /* yields 9*12=108                       */
 ratio   = 9 / 12              /* yields 9/12=0                         */
return(sum);
}
```

Floating Point vs Integer Divide

- There is a vast difference between an integer divide and a floating point divide. In an integer divide the result is truncated.
- C allows the assignment of an integer expression to a floating point variable. C will automatically do the conversion. Similarly, a floating point number can be assigned to an integer variable; the value will be truncated.
- The result of a division is integer only if both factors are integers or if the result is assigned to an integer variable. Otherwise it is floating point.

Expression	Result	Result Type
19 / 10	1	integer
19.0 / 10	1.9	floating point
19.0 / 10.0	1.9	floating point

Floating Point vs Integer Divide - Example

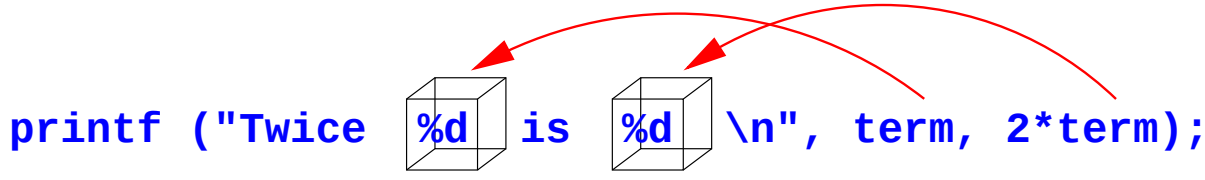
```
/*********************************************************
 * Question:                                             *
 *      Why does the following program print:            *
 *      "The value of 1/3 is 0.0" ?                      *
 *********************************************************/
#include <stdio.h>

float answer;   /* The result of our calculation */

int main()
{
    answer = 1/3;
    printf("The value of 1/3 is %f\n", answer);
    return (0);
}
```

Answer: 1 and 3 are both integers, so the problem is the integer divide where the result is truncated. The expression should be written as

```
answer = 1.0 / 3.0
```

The Function `printf`

- The standard function in C used to print something to the screen is `printf`, included in the library `<stdio.h>`.
- The standard form of the `printf` statement is `printf(format, expression-1, expression-2, ...)` where `format` is the string describing what to print, e.g.
- `printf("Twice %d is %d", term, 2*term);` where the special characters `%d` are the integer conversion specification. Their place is filled with the values of `expression-1` and `expression-2`; everything else is printed verbatim.



- You have to check that conversions and expressions match !

Escape Characters

Special characters or escape characters starting with '\' move the cursor or represent otherwise reserved characters.

Character	Name	Meaning
\b	backspace	move cursor one character to the left
\f	form feed	go to top of new page
\n	newline	go to the next line
\r	return	go to beginning of current line
\a	audible alert	'beep'
\t	tab	advance to next tab stop
\'	apostrophe	character '
\"	double quote	character "
\\	backslash	character \
\nnn		character number nnn (octal)

Format Statements

Besides %d there are other conversion specifications, here is an overview of the most important ones:

Conversion	Argument Type	Printed as
%d	integer	decimal number
%f	float	[-]m.dddddd (details below)
%X	integer	hex. number using A..F for 10..15
%c	char	single character
%s	char *	print characters from string until '\0'
%e	float	float in exp. form [-]m.ddddddde±xx

In addition, the precision and additional spaces can be specified:

%6d	decimal integer, at least 6 characters wide
%8.2f	float, at least 8 characters wide, two decimal digits
%.10s	first 10 characters of a string

Example for `printf`

Why does 2 + 2 = 1075031184 ? (on my laptop; results may vary)

```
#include <stdio.h>

/* Variable for computation results */
int answer;

int main()
{
    answer = 2 + 2;

    printf("The answer is %d\n");
    return (0);
}
```

Answer: The `printf` statement is trying to print a decimal number (%d), but no value is specified - so C makes one up. The proper statement would be

```
printf("The answer is %d\n", answer);
```

ASCII and beyond

- ASCII was defined in 1968 (i.e before most of you were born). It uses 7 bit and works well with American English, which is after all what it was meant to do.
- ASCII already doesn't work with Danish, French or German - but for this there is an extension to 8 bit called Latin1 (aka ISO-8859-1).
- `printf` will typically understand Latin1, so you can use `printf("One beer is 1.95 \xA3");` to print `One beer is 1.95 £`.
- If you're German (like me) you might want to print `'Schöne Grüße'`, which you can do like this: `printf("Sch\366ne Gr\374\337e\n");`
- Of course this still doesn't help you much if you're Thai or Armenian. Have a look at `www.unicode.org` for Unicode (ISO-10646), a 32 bit code for all the letters in the world.