A survival kit for C language

Ing. Davide Piccinini

Operative System
Academic year 2017/2018

logoUnipr.png

%d %i	integer, are different in scanf() (where using %i will interpret a number as hexadecimal if it's preceded by 0x, and octal if it's preceded by 0)
%u	unsigned int
%f %F	float and double, in normal (fixed-point) notation
%e %E	float and double, in standard form
%g %G	float and double, in either normal or standard form (whichever is more appropriate for its magnitude)
%x %X	unsigned integer as a hexadecimal number (x for lowercase or X for uppercase letters)
%o	unsigned int as octal
%s	null-terminated string (NOTE: null-character is '\0')
%c	char
%p	void * (pointer to void)
%a %A	float and double in hexadecimal notation, starting with 0x or 0X (a for lowercase or A for uppercase letters)

Table 1: Argument formats

`\n` is the newline character into strings.
`\t` is the tabular character.

Input

```
int printf("string with arguments formats", variables);
```

Its an output function used to print on the screen (STDOUT) text and variables. It take a string in which you can put one or more arguments format (or format placeholder) (Table **??**) and variables, separated by the string, that is enclosed in double quotes ("), and from each other by a comma (,).
NOTE: printf format strings are complementary to scanf format strings.

Example:

```
int old_year = 2017;
int year = 2018;
printf("This is the academic year \%i/\%i", old_year, year);
```

Output:

```
This is the academic year 2017/2018
```

The syntax for a format placeholder is:

`%[parameter][flags][width][.precision]type`

- **Flags field:**

-	Left-align the output of this placeholder. (The default is to right-align the output.)
+	Prepends a plus for positive signed-numeric types. positive = +, negative = -. (The default doesn't prepend anything in front of positive numbers.)
␣	Prepends a space for positive signed-numeric types. positive = , negative = -. This flag is ignored if the + flag exists. (The default doesn't prepend anything in front of positive numbers.)
0	When the 'width' option is specified, prepends zeros for numeric types. (The default prepends spaces.)
#	Alternate form: For g and G types, trailing zeros are not removed. For f, F, e, E, g, G types, the output always contains a decimal point. For o, x, X types, the text 0, 0x, 0X, respectively, is prepended to non-zero numbers.

- **The Width field:** specifies a minimum number of characters to output, and is typically used to pad fixed-width fields in tabulated output, where the fields would otherwise be smaller, although it does not cause truncation of oversized fields.

- **The Precision field:** usually specifies a maximum limit on the output, depending on the particular formatting type. For floating point numeric types, it specifies the number of digits to the right of the decimal point that the output should be rounded. For the string type, it limits the number of characters that should be output, after which the string is truncated.
- **The type field:** see Table **??**.

`printf.png`

Output

```
int scanf("arguments formats", &variables);
```

Its an input function used to take arguments from user input (STDIN). It take a string composed of one or more arguments format (Table **??**) and one or more addresses of variables, separeted by a comma (,), (this addresses are obtained with variable name preceded from &).

NOTE: to read a string you do not have to use & because a string (an array of char) its already an address, to the head of the array.

Example:

```
char name[100];
unsigned int age;
scanf("\%s \%u", name, age);
```

Struct & Typedef

```
struct <name> {
     variables
}
```

This is a structure that can contain one (useless) or more parameters. Example:

```c
struct rectangle{
  int s1;
  int s2;
};
```

Use:

```c
int main(int argc, char **argv) {
  ...
  struct rectangle r1;
  ...
  return 0;
}
```

```
typedef <existent data type> <name>
```

Is a keyword used to rename a existent data type or structure.

NOTE: with `typedef` we can omit the keyword "*struct*" when we use structures as type.

Example:

```c
typedef int side;

typedef struct {
  side s1;
  side s2;
}rectangle;
```

Use:

```c
int main(int argc, char **argv) {
  ...
  rectangle r1;
  ...
  return 0;
}
```

Socket & Send/Recive

Sockets are a way of connecting two nodes on a network to communicate with each other. To create a socket and to connect with them we have to implement different methods (Figure 1**??**).

socket.png

Figure 1: Server-Client socket comunication

```
int <sockfd name> = socket(<domain>, <type>, <protocol>)
```

Function used to create a new file descriptor of a socket

- **domain** → usually AF_INET (for IPv4) or AF_INET6 (for IPv6).
- **type** → usually SOCK_STREAM (with TCP) or SOCK_DGRAM (with UDP.)
- **protocol** → usually 0, this cause socket() to use an unspecified default protocol appropriate for the requested socket type.

```
int setsockopt(int <sockfd>, int <level>, int <optname>,
const void *<optval>, socklen_t <optlen>);
```

- **level** → SOL_SOCKET.
- **optname** → usually SO_REUSEADDR, to reuse address.
- ***optval** → usually an address of a variable set at 1 (*remember: address of a variable is passed by preceding the name by an* &).
- **optlen** → is sizeof(<name of the variable passed to optval, without &>).
  NOTE: sizeof(<argument>) is a function that return the size of the passed argument

```
int bind(int <sockfd>, const struct sockaddr *<addr>,
socklen_t <addrlen>);
```

- ***addr** → usually we create a variable of type "`const struct sockaddr`", in which we insert some parameters related to the socket (lool at the example below), and pass his address using &<name of the variable>.
- **addrlen** → is sizeof(<name of the variable passed to addr, without &>).

```
int listen(int <sockfd>, int <backlog>);
```

- **backlog** → a number that defines the maximum length to which the queue of pending connections for <sockfd> may grow.

```
int <new_socket name> = accept(int <sockfd>, struct sockaddr
*<addr>, socklen_t *<addrlen>);
```

```
int connect(int <sockfd>, struct sockaddr *<addr>, socklen_t
<addrlen>);
```

Example below!

Server

```c
#include <unistd.h>
#include <sys/socket.h>  //used for all the functions
#include <sys/types.h>   //related with socket
#include <stdio.h>
#include <netdb.h>  //used for struct sockaddr_in

#define PORT 8000

int main(int argc, char **argv) {
  struct sockaddr_in addr;
  int newsock;
  int opt = 1;
  socklen_t addr_len;
  ...

  int servfd = socket(AFINET, SOCK_STREAM, 0);
  if(servfd < 0) {
    perror("Error! Socket failed.\n");
    exit(-1);
  }

  if(setsockop(servfd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt)
      ) < 0) {
    perror("Error! Setsockop failed.\n");
    exit(-1);
  }

  bzero(&addr, sizeof(addr)); //Used to zero the address
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port = htons(PORT);

  addr_len = sizeof(addr);

  if(bind(servfd, (struct sockaddr *)&addr, sizeof(addr))<0) {
    perror("Error! Binding failed.\n");
    exit(-1);
  }

  if(listen(servfd, 5) < 0) {
    perror("Error! Binding failed.\n");
    exit(-1);
  }

  for(;;) {
    newsock = accept(servfd, (struct sockaddr *)&addr, &addr_len);
    if(newsock < 0) {
      perror("Error! Accept failed.\n");
      exit(-1);
    }

    //TODO, send and recive, ...
  }
  ...
  return 0;
}
```

Client

```c
#include <unistd.h>
#include <sys/socket.h>  //used for all the functions
#include <sys/types.h>   //related with socket
#include <stdio.h>
#include <netdb.h>   //used for struct hostent and gethostbyname()

#define PORT 8000

int main(int argc, char **argv) {
  struct sockaddr_in serv_addr;
  struct hostent* server;
  ...

  server = gethostbyname(host_name);
  if (server == 0) {
    perror("Error resolving local host!\n");
    exit(1);
  }

  int clientfd = socket(AFINET, SOCK_STREAM, 0);
  if(clientfd < 0) {
  perror("Error! Socket failed.\n");
  exit(-1);
  }

  bzero(&serv_addr, sizeof(serv_addr)); //Used to zero the address
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = ((struct in_addr *)(server->h_addr))
      ->s_addr;;
  serv_addr.sin_port = htons(PORT);

  if(connect(clientfd, (struct sockaddr *)&serv_addr, sizeof(
      serv_addr)) < 0) {
    perror("Error! Connection failed.\n");
    exit(-1);
  }

  for(;;) {
    //TODO, send and recive, ...
  }
  ...
  return 0;
}
```

```
ssize_t send(int <sockfd>, const void *<buf>, size_t <len>,
int <flags>);
```

- ***buf** → address of a variable that contain the messages taht must be send to the client/server (remember: is passed by preceding the name by an &).
- **len** → sizeof(<name of the variable passed to buf, without &>).
- **flags** → usually 0 (in this way send() is equivalent to a write()).

```
ssize_t recv(int <sockfd>, const void *<buf>, size_t <len>,
int <flags>);
```

- **flags** → usually 0 (in this way send() is equivalent to a read()).

Useful Linux command and concepts

`/` - root directory.

`/bin` and `/usr/bin` - essential user binary directory and user binaries.

The `/bin` and `/usr/bin` directories contains, respectively, the essential and non-essential user binaries (programs) that must be present when the system is mounted in single-user mode.

`/home` – home folders.

The `/home` directory contains a home folder for each user.

NOTE: `/home/<user name>` and ∼ are the same things.

`..` is the parent directory, the previous folder into the Linux hierarchy.

`.` is the current directory.

`?` is a meta-character used, in Linux, to indicate one and only one general character.

Example:

`rm exercise?.c` ⟶ delete every file, in the actual directory, with a name composed by "*exercise*" - *one character* - "*.c*"

`*` is a meta-character used, in Linux, to indicate one **or more** general character.

Example:

`rm exercise* *.txt` ⟶ delete every file, in the actual directory, that start with "*exercise*" and every file with extension "*.txt*"

```
$ man <command/function>
```

Is a utility that show a page of the Linux manual about the command or function passed to it as an argument.

NOTE: use "`man man`" to obtain more information about man command.

```
$ cd <path from where you are to where you want to go>
```

Is a command used to navigate through Linux folders.

NOTE: "`cd ..`" allow you to go back to the parent dyrectory.
NOTE2: "`cd`" allow you to go back directly into the "∼" directory.

```
$ ls
```

Is a command that show a list of folders and files present in the current folder.

NOTE: "`ls -a`" used to also show you the hidden folders and files.
NOTE2: "`ls -l`" used to show more information about folders and files.

How to compile and run a C-program in Linux

```
$ gcc -o -Wall <executable name> <file name>.c
```

NOTE: "`-Wall`" is a warning option, which allows you to have a much finer control of the warnings.

```
$ ./<executable name>
```

Allow you to run the program.