

**1.** O vector `texto` contém uma *string*. Complete o programa abaixo de forma a fazer o envio dessa *string* (excluindo o `'\0'`) para o outro extremo da ligação associada ao descritor de *socket* `s`.

```
char texto[100];
int s = socket(PF_INET, SOCK_STREAM, 0);
//assuma que server_address está correctamente iniciada
connect(s, &server_address, sizeof(server_address));
```

**Sol1:**

```
write(s, texto, strlen(texto));
```

**Sol2:**

```
FILE *fp = fdopen(s, "w");
fputs(texto, fp);
```

**2.** Pretende-se enviar o 3º elemento do vector `valores` (definido abaixo) através de um *socket* do tipo `SOCK_STREAM`. Assumindo que o descritor do *socket* é a variável `sock1` e que a ligação já está activa, complete o programa.

```
float valores[100];
```

**Sol1:**

```
write(sock1, valores+2, sizeof(float));
//nota: valores+2 é equivalente a &(valores[2])
```

**Sol2:**

```
FILE *fp = fdopen(sock1, "w");
fwrite(valores+2, sizeof(float), 1, fp);
```

**3.a)** Pretende-se enviar o campo `idade` da variável `reg1` através de um *socket* do tipo `SOCK_STREAM`. Assumindo que o descritor do *socket* é a variável `sock1` e que a ligação já está activa, complete o programa.

```
typedef struct {
    char nome[100];
    int idade;
    float peso;
} registo_t;

registo_t reg1;
```

**Sol1:**

```
write(sock1, &(reg1.idade), sizeof(reg1.idade));
//nota: &(reg1.idade) é equivalente a &reg1.idade
```

**Sol2:**

```
FILE *fp = fdopen(sock1, "w");
fwrite(&(reg1.idade), sizeof(reg1.idade), 1, fp);
```

**3.b)** Utilizando o mesmo *socket* (assuma que `sock1` é uma variável global), complete a função de modo a fazer o envio do campo `peso` da variável apontada por `preg`.

```
void envia(registo_t *preg) {
```

---

Sol1:

```
write(sock1, &(preg->peso), sizeof(preg->peso));  
// (preg->peso) é equivalente a (*preg.peso)
```

Sol2:

```
FILE *fp = fdopen(sock1, "w");  
fwrite(&(preg->peso), sizeof(preg->peso), 1, fp);
```

4. Considere o programa abaixo, onde se pretende fazer a leitura de 10 bytes a partir do *socket* *sock1*, correspondente a uma comunicação *SOCK\_STREAM* activa, e enviar, através do mesmo *socket*, o valor da variável *estado*. Identifique os dois erros do programa e apresente uma solução correcta.

```
1: char *ptr;  
2: int estado = 1;  
3: FILE *fp = fdopen(sock1, "r+");  
4: fread(ptr, 10, 1, fp);  
5: write(sock1, estado, sizeof(estado));
```

Sol:

Na linha 4, tenta-se guardar 10 bytes num apontador não iniciado. Possíveis soluções seriam alterar a linha 1 para:

```
char ptr[10];
```

ou

```
char *ptr = malloc(10);
```

Na linha 5, existe um erro na passagem do 2º parâmetro. Deveria ser:

```
write(sock1, &estado, sizeof(estado));
```

5. Considere o programa abaixo, onde se pretende fazer a leitura de 2 inteiros a partir do *socket* *sock1* e guardar a sua soma no ficheiro "resultado.txt". Identifique os dois erros do programa e apresente uma solução correcta.

```
1: int dados[2];  
2: int result;  
3: int sock1 = socket(PF_INET, SOCK_STREAM, 0);  
4: //assuma que serv_address está correctamente iniciada  
5: connect(sock1, &serv_address, sizeof(serv_address));  
6: fread(dados, 10, 1, sock1);  
7: FILE *fp = fopen("resultado.txt", "w");  
8: result = dados[0]+dados[1];  
9: fwrite(result, sizeof(result), 1, fp);
```

Sol:

Na linha 6, o *fread* não pode ser usado com um "file descriptor" (*sock1*). Além disso, o tamanho de 2 inteiros é  $2 * \text{sizeof}(\text{int})$  e não 10:

```
FILE *fps = fdopen(sock1, "r");
```

```
fread(dados, sizeof(int), 2, fps);  
//ou fread(dados, 2*sizeof(int), 1, fps);  
/* ou até, mas tecnicamente uma solução pior:  
fread(dados, sizeof(int), 1, fps);  
fread(dados+1, sizeof(int), 1, fps);  
*/
```

Na linha 9, existe um erro na passagem do 1º parâmetro. Deveria ser:

```
fwrite(&result, sizeof(result), 1, fp);
```

6. Pretende-se receber um valor do tipo `double` a partir de um *socket* (variável `sock1`) do tipo `SOCK_STREAM` que se encontra já num estado pronto para transferência de dados. Complete o programa de modo a que o valor recebido fique armazenado na variável `valor`.

```
double valor;
```

Sol:

```
FILE *fp = fdopen(sock1, "r");
fread(&valor, sizeof(double), 1, fp);
```

7. Pretende-se receber um valor do tipo `double` a partir de um *socket* (variável `sock1`) do tipo `SOCK_STREAM` que se encontra já num estado pronto para transferência de dados. Complete a função de modo a que o valor recebido fique armazenado no endereço apontado por `pvalor` (passagem por referência).

```
void leitura1(int sock1, double *pvalor) {
```

Sol:

```
FILE *fp = fdopen(sock1, "r");
fread(pvalor, sizeof(double), 1, fp);
```

8. Complete o programa abaixo de forma a fazer a leitura de uma variável do tipo `registo_t` a partir do *socket*. Os dados recebidos deverão ser armazenados na variável `r`.

```
registo_t r;
int s = socket(PF_INET, SOCK_STREAM, 0);
//assuma que serv_address está correctamente iniciada
connect(s, &server_address, sizeof(server_address));
```

Sol:

```
FILE *fp = fdopen(s, "r");
fread(&r, sizeof(r), 1, fp);
```

9. Complete o programa abaixo de forma a fazer a leitura de uma sequência de 100 valores do tipo `int` a partir do *socket*. Os dados recebidos deverão ser armazenados na variável `vector`.

```
int vector[100];
int s = socket(PF_INET, SOCK_STREAM, 0);
//assuma que serv_address está correctamente iniciada
connect(s, &server_address, sizeof(server_address));
```

Sol:

```
FILE *fp = fdopen(s, "r");
fread(vector, sizeof(int), 100, fp);
```

10. Complete o programa abaixo de forma a fazer a leitura de uma sequência de 30 valores do tipo `int` a partir do *socket*. Os dados recebidos deverão ser armazenados a partir do 10º elemento da variável `vector`.

```
int vector[100];
int s = socket(PF_INET, SOCK_STREAM, 0);
//assuma que serv_address está correctamente iniciada
connect(s, &server_address, sizeof(server_address));
```

Sol:

```
FILE *fp = fdopen(s, "r");
fread(&(vector[9]), sizeof(int), 30, fp);
```

11. Classifique as seguintes afirmações como verdadeiras (V) ou falsas (F).

Um <i>socket</i> SOCK_STREAM pode ser utilizado para transferência de dados antes de se efectuar o <i>connect</i> .	F
Um <i>socket</i> SOCK_STREAM poderá ser utilizado para enviar dados para vários destinos.	F
Um <i>socket</i> SOCK_STREAM apenas pode receber dados de uma única fonte.	V
Uma comunicação SOCK_STREAM é fiável.	V
Numa comunicação SOCK_STREAM, os dados poderão ser recebidos em duplicado ou fora de ordem.	F
Numa comunicação SOCK_STREAM, os dados são enviados sequencialmente sem qualquer tipo de delimitação entre escritas sucessivas.	V
Assumindo que cada envio corresponde a um volume de dados inferior ao tamanho dos pacotes da rede de dados, então cada leitura efectuada a um <i>socket</i> SOCK_STREAM (com <i>buffer</i> adequado) obtém todos os dados correspondentes a uma (e só uma) escrita no <i>socket</i> do outro extremo.	F
Cada <i>read</i> de um <i>socket</i> SOCK_STREAM retorna os dados correspondentes a um (e só um) <i>write</i> do outro extremo.	F
Numa comunicação SOCK_STREAM, os dados só podem ser enviados num sentido de cada vez ( <i>half-duplex</i> ).	F
Um <i>socket</i> do tipo SOCK_STREAM pode ser usado para enviar dados mesmo que ainda tenha dados prontos para leitura ou a receber do outro extremo.	V
A função <i>sendto</i> é útil para comunicações do tipo SOCK_STREAM	F
Os mesmos dados poderão ser lidos a partir do <i>socket</i> mais do que uma vez, usando a função <i>lseek</i> .	F
Os <i>sockets</i> do tipo SOCK_STREAM são a escolha mais adequada para a transmissão de áudio em tempo real através da <i>internet</i> .	F
Numa comunicação com <i>sockets</i> SOCK_STREAM, apenas o servidor necessita de usar a função <i>bind</i> .	V
Um <i>socket</i> do tipo SOCK_DGRAM pode ser utilizado para transferência de dados sem se efectuar um <i>connect</i> , usando para tal a função <i>sendto</i> .	V
Um <i>socket</i> do tipo SOCK_DGRAM poderá ser utilizado para enviar dados para diferentes destinatários	V
A função <i>connect</i> pode ser usada com <i>sockets</i> do tipo SOCK_DGRAM para estabelecer um destino <i>default</i> para os dados.	V
Assumindo que cada envio corresponde a um volume de dados inferior ao tamanho dos pacotes da rede de dados, então cada leitura efectuada a um <i>socket</i> SOCK_DGRAM (com <i>buffer</i> adequado) obtém todos os dados correspondentes a uma (e só uma) escrita no <i>socket</i> do outro extremo.	V
Os <i>socket</i> do tipo SOCK_DGRAM oferecem comunicações fiáveis.	F
Numa comunicação do tipo SOCK_DGRAM, os dados poderão ser recebidos em duplicado ou fora de ordem.	V
Numa comunicação do tipo SOCK_DGRAM, os dados são enviados sem qualquer tipo de delimitação entre escritas sucessivas.	F
Numa comunicação do tipo SOCK_DGRAM, ambos os extremos da comunicação poderão enviar dados simultaneamente ( <i>full-duplex</i> ).	V

12. Em relação à utilização da função **bind** em comunicações do tipo **SOCK\_STREAM**, classifique as seguintes afirmações como verdadeiras (V) ou falsas (F).

Serve para aceitar ligações	F
É, em geral, usada para atribuir um endereço, escolhido pelo utilizador ou programador, ao <i>socket</i> <sup>1</sup> *	V
Deverá ser usada nos servidores	V
Deverá ser usada nos clientes	F
Só é usada para a família de protocolos <i>Internet Protocol</i> (IP)	F
É uma função não-bloqueante	V
Exceptuando situações de erro, retorna um novo socket	F

13. Em relação à utilização da função **accept**, classifique as seguintes afirmações como verdadeiras (V) ou falsas (F).

Em comunicações do tipo <b>SOCK_STREAM</b> , deverá ser usada pelo cliente.	F
É uma função opcional para comunicações do tipo <b>SOCK_STREAM</b> .	F
Nunca é usada com <i>sockets</i> do tipo <b>SOCK_STREAM</b>	F
Em comunicações do tipo <b>SOCK_DGRAM</b> , pode ser usada pelo receptor, para evitar o uso da função <b>recvfrom</b> .	F
Nunca é usado com sockets do tipo <b>SOCK_DGRAM</b>	V
Só é usada para a família de protocolos <i>Internet Protocol</i> (IP).	F
Por defeito, é uma função bloqueante; aguarda que sejam feitas ligações.	V
Por defeito, no caso de <u>não</u> haver ligações pendentes, retorna imediatamente (não-bloqueante).	F
Deverá ser-lhe passado, como argumento, o endereço do <i>socket</i> remoto.	F
No caso de haver ligações pendentes, retorna um descritor de socket, que deverá ser usado para a transferência de dados com o outro extremo.	V
O primeiro argumento da função <b>accept</b> deverá ser um descritor de <i>socket</i> ao qual se aplicou a função <b>listen</b> .	V
É usada para atribuir um endereço, escolhido pelo utilizador ou programador, ao <i>socket</i>	F

---

1 No caso da família IP, é possível solicitar uma atribuição automática do porto, preenchendo o campo `sin_port` com 0.

14. Em relação à utilização da função **connect**, classifique as seguintes afirmações como verdadeiras (V) ou falsas (F).

Em comunicações do tipo SOCK_STREAM, deverá ser usada pelo cliente.	V
Em comunicações do tipo SOCK_STREAM, deverá ser usada pelo servidor.	F
É uma função bloqueante; aguarda que o outro extremo faça o <b>accept</b> ou uma leitura.	F
Deverá receber como argumento o endereço do <i>socket</i> remoto.	V
Em comunicações do tipo SOCK_DGRAM, pode ser usada pelo emissor, para evitar o uso da função <b>sendto</b>	V
Só é usada para a família de protocolos <i>Internet Protocol</i> (IP).	F
Exceptuando situações de erro, retorna um novo <i>socket</i>	F
Serve para aguardar ligações	F
É uma função opcional para comunicações do tipo SOCK_STREAM	F

Comentários às questões 11-14:

- As afirmações “*Numa comunicação SOCK\_STREAM, os dados são enviados sequencialmente sem qualquer tipo de delimitação entre escritas sucessivas*”, “*Assumindo que cada envio corresponde a um volume de dados inferior ao tamanho dos pacotes da rede de dados, então cada read() efectuado a um socket SOCK\_STREAM (com buffer adequado) obtém todos os dados correspondentes a um (e só um) envio do outro extremo*” e “*Cada read de um socket SOCK\_STREAM retorna os dados correspondentes a um (e só um) write do outro extremo*” referem-se ao mesmo aspecto. Esse comportamento dos sockets SOCK\_STREAM causa algumas dificuldades na leitura dos dados, daí ser preferível converter o descritor de *socket* num *stream* (através do `fdopen`). Note-se que, neste caso, as afirmações referem-se a leituras directas ao descritor de *socket* (`read`, `recv`, `recvfrom`)
- Comunicação fiável implica que os dados não são recebidos nem fora de ordem nem em duplicado.
- Uma ressalva acerca da fiabilidade: os sockets PF\_INET/SOCK\_STREAM são fiáveis assumindo que não existem “ataques” a nível da rede de dados que interceptem e corrompam, de forma inteligente, os pacotes de dados (e.g., “man-in-the middle”).
- Das funções `socket`, `bind`, `listen`, `accept`, `connect`, a única bloqueante (BLOCKING) é a `accept`. Note-se que se refere ao comportamento *default*, uma vez que é possível configurar os sockets de modo a que as operações sobre ele não sejam bloqueantes.
- Os *sockets* não permitem acesso aleatório aos dados (`lseek`, `fseek`, etc.). Cada byte é “consumido” a partir do momento em que é lido.
- Os tipos de sockets estudados permitem sempre comunicações *full-duplex*.
- A escolha do tipo de comunicação (SOCK\_STREAM, SOCK\_DGRAM) é independente da família de protocolos. Por exemplo, para um socket PF\_LOCAL, SOCK\_STREAM, o protocolo usado não será o TCP.

15. Assumindo que não se pretende ativar nenhuma opção especial para a comunicação via *sockets* (parâmetro *flags*), classifique as seguintes afirmações como verdadeiras (V) ou falsas (F).

A função <b>recvfrom</b> torna-se necessária para a receção de dados num <u>servidor</u> :	
a) Sempre que o cliente usa a função <b>sendto</b> .	F
b) Em comunicações do tipo SOCK_STREAM, antes de se efetuar o <b>accept</b> .	F
c) Em comunicações do tipo SOCK_STREAM, de forma a permitir o envio de resposta a clientes cujo endereço não foi registado na chamada à função <b>accept</b> .	F
d) Em comunicações do tipo SOCK_STREAM, mesmo que não se pretenda conhecer o endereço do cliente.	F
e) Em comunicações do tipo SOCK_STREAM, quando se pretende seleccionar o cliente do qual serão recebido dados.	F
f) Em comunicações do tipo SOCK_DGRAM, de forma a permitir o envio de resposta a clientes sem endereço previamente conhecido.	V
g) Em comunicações do tipo SOCK_DGRAM, mesmo que não se pretenda conhecer o endereço do cliente.	F
h) Em comunicações do tipo SOCK_DGRAM, quando se pretende seleccionar o cliente do qual serão recebido dados.	F

16. Em relação à utilização da função **sendto**, classifique as seguintes afirmações como verdadeiras (V) ou falsas (F).

a) É essencial para comunicações do tipo SOCK_STREAM.	F
b) Em comunicações do tipo SOCK_STREAM, permite enviar mensagens para diferentes destinos através de um mesmo <i>socket</i> .	F
c) Em comunicações do tipo SOCK_DGRAM, permite enviar mensagens para diferentes destinos através de um mesmo <i>socket</i> .	V
d) Em comunicações do tipo SOCK_DGRAM, deverá receber sempre como argumento o endereço do <i>socket</i> remoto.	V
e) Em comunicações do tipo SOCK_STREAM, pode ser usada pelo emissor, para evitar o uso da função <b>connect</b> .	F
f) Só é usada para a família de protocolos <i>Internet Protocol</i> (AF_INET).	F
g) O tamanho máximo permitido para as mensagens enviadas em sockets do tipo SOCK_DGRAM é, em geral, limitado.	V
h) Para <i>sockets</i> AF_INET/SOCK_DGRAM, retorna o número de <i>bytes</i> recebidos no <i>socket</i> de destino.	F
i) Requer a utilização prévia da função <b>bind</b> no mesmo programa.	F

17. Complete o programa abaixo de forma a fazer a receção de uma variável do tipo `registo_t` a partir da ligação estabelecida (os dados deverão ser armazenados na variável `r`). Os dados recebidos deverão ser armazenados na variável `r`.

```
registo_t r;
int s = socket(PF_INET, SOCK_STREAM, 0);
(...)
int ns = accept(s, NULL, 0);
```

**Sol:**

```
FILE *fp = fdopen(ns, "r");
fread(&r, sizeof(r), 1, fp);
```

**18.** Complete o programa abaixo de forma a fazer a recepção de uma sequência de 30 valores do tipo `int` a partir da ligação estabelecida. Os dados recebidos deverão ser armazenados na variável `vector`.

```
int vector[30];
int s = socket(PF_INET, SOCK_STREAM, 0);
(...)
int ns = accept(s, NULL, 0);
```

Sol:

```
FILE *fp = fdopen(ns, "r");
fread(vector, sizeof(int), 30, fp);
```

**19.** Complete o programa abaixo de forma a fazer a leitura de uma sequência de 6 caracteres a partir da ligação estabelecida. Os dados recebidos deverão ser armazenados no campo `codigo` da variável `reg1`.

```
typedef struct {
    char codigo[6];
    char designacao[100];
    float peso;
} registo_t;
registo_t reg1;

int s = socket(PF_INET, SOCK_STREAM, 0);
(...)
int ns = accept(s, NULL, 0);
```

Sol:

```
FILE *fp = fdopen(ns, "r");
fread(reg1.codigo, 1, 6, fp);
```

**20.** Considere o seguinte extrato de um programa:

```
1: int main(){
2:     struct sockaddr_in sa;
3:     double valor = 0;
4:
5:     int s = socket(AF_INET, SOCK_STREAM, 0);
6:
7:     sa.sin_family = AF_INET;
8:     sa.sin_addr.s_addr = INADDR_ANY;
9:     sa.sin_port = htons(1024);
10:    bind(s, (struct sockaddr *) &sa, sizeof(sa));
11:
12:    listen(s, 5);
13:
14:    while(1) {
15:        printf("valor = %.2f\n", valor);
16:        ns = accept(s, NULL, 0);
17:        //completar
18:    }
19:    return 0;
20: }
```

**a)** Indique qual a finalidade da chamada à função `bind` (linha 10).

**b)** Complete o programa de forma a que, logo após o estabelecimento de uma ligação, seja efetuada a seguinte sequência de ações (pode omitir as verificações de erro):

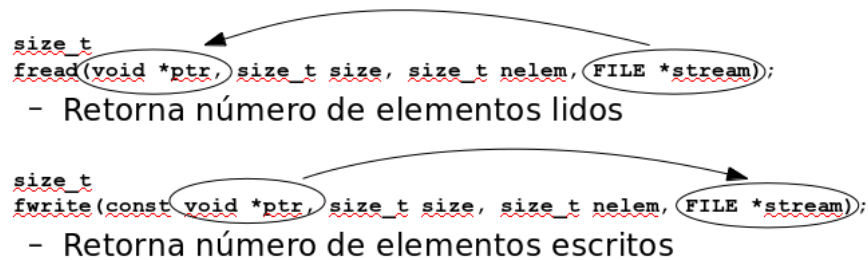
- Recepção, a partir da ligação estabelecida, de um valor do tipo `double` (em formato binário), que deverá ser guardado na variável `valor`.
- Envio, através da mesma ligação, do valor recebido, em formato de texto (*string* de formato `"%.2f\n"`).
- Escrita do valor recebido (*double* em formato binário) no final do ficheiro `"dados.dat"`.
- Fecho da ligação.



a) A finalidade da chamada à função `bind` é associar um endereço (ou nome), neste caso do tipo `AF_INET`, ao socket `s` (criado na linha 5). Mais especificamente, pretende-se que o socket fique associado ao porto 1024 de todos os endereços IP da máquina (`INADDR_ANY`).

b)

```
FILE *fps = fdopen(ns, "r+");
fread(&valor, sizeof(double), 1, fps);
fprintf(fps, "%.2f\n", valor);
fclose(fps);
FILE *fpf = fopen("dados.dat", "a");
fwrite(&valor, sizeof(double), 1, fpf);
fclose(fpf); // fecho do ficheiro, caso contrário não há garantia que o valor seja escrito
```



`size_t fread(void *ptr, size_t size, size_t nelem, FILE *stream);`  
- Retorna número de elementos lidos

`size_t fwrite(const void *ptr, size_t size, size_t nelem, FILE *stream);`  
- Retorna número de elementos escritos

21. Complete o programa abaixo de forma a fazer a leitura de caracteres do socket `s` até que a ligação seja terminada pelo outro extremo. Os dados recebidos deverão ser armazenados na variável `texto`. Assuma que o número de caracteres recebidos é inferior a 256.

```
char texto[256];
int s = socket(PF_INET, SOCK_STREAM, 0);
//assuma que serv_address está correctamente iniciada
connect(s, &server_address, sizeof(server_address));
```

**Sol1:**

```
int i = 0;
int c;
FILE *fp = fdopen(s, "r");
while(1) {
    c = fgetc(fp);
    if(c==EOF)
        break;
    texto[i] = c;
    i++;
}
```

**Sol2:**

```
FILE *fp = fdopen(s, "r");
fread(texto, 1, 255, fp); // neste caso, o valor de retorno do fread indica o nº de caracteres
// recebidos
```

22. Complete o programa abaixo de forma a fazer a recepção de uma sequência de caracteres terminada em `\n` a partir da ligação estabelecida. Os dados recebidos deverão ser armazenados na variável `texto`. Assuma que o número de caracteres recebidos é inferior a 100.

```
int texto[100];
int s = socket(PF_INET, SOCK_STREAM, 0);
//assuma que serv_address está correctamente iniciada
connect(s, &server_address, sizeof(server_address));
```

**Sol:**

```
FILE *fp = fdopen(s, "r");
fgets((char *) texto, 100, fp);
```

```
/* Nota: aceitaram-se como correctas as respostas que assumiam que o vector texto era do tipo
char, ou seja: */

char texto[100];
FILE *fp = fdopen(s, "r");
fgets(texto, 100, fp);
```