```
/*================================ Servidor UDP
===========================

  Este servidor recebe inteiros de clientes UDP até o temporizador acabar ou atingir o valor máximo de
clientes. Todos os valores múltiplos de 3 e pares são contados. No final o servidor constrói uma
mensagem onde inclui o resultado e envia-o a todos os clientes.

  ========================================================
=========================*/

#include <stdio.h>

#include <winsock.h>


#define SERV_UDP_PORT 60001

#define MAX_MSGS 5

#define TIMEOUT 60000; //msec

#define MAX_RESPOSTA 100


void Abort(char *msg);


int main( int argc , char *argv[] )

{

DWORD timeout;

WSADATA wsaData;

SOCKET s;

int iResult, nbytes, tam;

int i, continua;

struct sockaddr_in serv_addr , cli_addr;


// Questão 1 -  Esta e' a tabela onde são guardadas as origens das mensagens.

//    Complete a sua declaracao com o tipo mais adequado.


struct sockaddr_inorigensMsgs[MAX_MSGS];


char resposta[MAX_RESPOSTA];

int valorRecebido, nMensagensRecebidas, contador;




iResult = WSAStartup(MAKEWORD(2,2), &wsaData);

if (iResult != 0) {
```

```
printf("WSAStartup failed: %d\n", iResult);
getchar();
exit(1);
}

// Questão 2 - Crie um socket UDP
if((s = socket(PF_INET, SOCK_DGRAM, 0)) == INVALID_SOCKET)
Abort("Impossibilidade de abrir socket");

memset( (char*)&serv_addr, 0, sizeof(serv_addr) );
serv_addr.sin_family = AF_INET;
serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
// Questão 3 - Preencha o campo relativo ao porto local pretendido.
serv_addr.sin_port =htons(SERV_UDP_PORT);

// Qestão 4 - Associe o socket ao porto pretendido recorrendo 'a estrutura serv_addr.
if(bind(s , (struct sockaddr *)&serv_addr, sizeof(serv_addr)) ==
SOCKET_ERROR){

// Questão 5 - Caso tenha ocorrido um erro, associe o socket a um porto automatico
//   (i.e., automaticamente atribuido).
serv_addr.sin_port = htons(SERV_UDP_PORT);

if(bind(s, (struct sockaddr *)&serv_addr , sizeof(serv_addr)) == SOCKET_ERROR){
Abort("Impossibilidade de registar-se para escuta");
}

// Questão 6 - Obtenha o valor do porto que foi automaticamente atribuido.
tam = sizeof(cli_addr);
if(getsockname(s, &serv_addr, &tam) != SOCKET_ERROR){

// Questão 7 -  Visualize o porto automatico.
printf("Porto local atribuido automaticamente: %d\n", serv_addr.sin_port);


}
```

}

while(1){

fprintf(stderr,"\n<Servidor> Esperando nova vaga de valores...\n\n");

nMensagensRecebidas = 0;

total = 0;

continua = 1;

while(nMensagensRecebidas < MAX_MSGS && continua){

// **Questão 8 - Aguarde pela recepcao de um datagrama UDP no socket s.**

// **O conteudo deve ser colocado na variavel valorRecebido.**

tam = sizeof(cli_addr);

nbytes=recvfrom(s, **(char *)valorRecebido**, **sizeof(valorRecebido)** , 0 , (struct sockaddr *)&cli_addr , &tam);

if(nbytes == SOCKET_ERROR){

// **Questão 9- Verifique se ocorreu algum timeout.**

if(**WSAGetLastError() == WSAETIMEDOUT**)

continua = 0;

else

Abort("Erro na recepcao de datagramas");

}else{

// **Questão 10 - Guarde as coordenadas da origem da mensagem na tabela**

// origensMsgs no indice com valor igual a nMensagensRecebidas.

origensMsgs[nMensagensRecebidas] = cli_addr;

nMensagensRecebidas++

// **Questão 11 - Conte os valores pares e múltiplos de 3**

if(**(valorRecebido % 2) == 0 && (valorRecebido % 3) == 0**) contador **+= 1** ;

printf("<Servidor> Valor recebido: %f\n", valorRecebido);

if(nMensagensRecebidas == 1){

//**Questão 12 - Active o timeout de recepcão com um valor definido pela constante TIMEOUT**

timeout = **(dword) TIMEOUT**;

setsockopt(s, SOL_SOCKET, **SO_RCVTIMEO, (char *)&timeout**, **sizeof(timeout)**);

}

}

}

// **Questão 13 - Construa a resposta de acordo com o solicitado no enunciado.**

// **Coloque a mensagem de texto resultante na variavel/string reposta**

sprintf(resposta, "N. de mensagens recebidas: %d ; Número de valores múltiplos de 3 e pares : %d", nMensagensRecebidas, total);

for(i=0; i<nMensagensRecebidas; i++){

// **Questão 14 - Envie o conteudo da variavel resposta a cada um dos destinos presentes na tabela origensMsgs.**

nbytes = sendto(s, resposta , strlen(resposta), 0 , (struct sockaddr *)**&origensMsgs**,sizeof(**&origensMsgs[i]**));

if (nbytes == SOCKET_ERROR){

printf("\n<Servidor> Erro %d ao reenviar a mensagem 'a origem no indice %d da tabela\n", WSAGetLastError(), i);

}

}

```
}

}

/*________________________________ Abort________________________________________
  Mostra uma mensagem de erro e o código associado ao ultimo erro com Winsocks.
  Termina a aplicacao com "exit status" a 1 (constante EXIT_FAILURE)
________________________________________________________________________________*/

void Abort(char *msg)
{
fprintf(stderr,"<Servidor> Erro fatal: <%s> (%d)\n",msg, WSAGetLastError());
exit(EXIT_FAILURE);
}
```