

<b>Estudante:</b>	<b>Matrícula:</b>
-------------------	-------------------

**Instruções:**

- Este exame possui 3 página(s). Verifique se sua cópia do exame está completa.
- Esta prova é sem consulta:** você não tem permissão para consultar o livro texto, suas notas de aula, a Internet, seus colegas ou quaisquer outras pessoas ou fontes para concluir o exame.
- Se você acredita que alguma pergunta esteja subespecificada, anote as suposições que você teve que fazer para chegar a sua resposta e justifique-as como parte de sua resposta à pergunta.
- Lembre-se de indentar o código** de maneira apropriada e atente-se a organização, clareza e legibilidade do código!

Nesta prova, vamos construir um software para gerenciar um banco. Os dados de cada cliente estarão armazenados em um arquivo CSV que obedece o seguinte formato (cada linha corresponde a um cliente):

```
1 Nome;Saldo;Limite;Ativo 1;Ativo 2;Ativo 3
2 Shakira;400.10;4000.00;A440.00;F7000.12;T300.00
3 David Guetta;1000.10;900.00;A00.44;T120.00;F10540.00
4 Jose Rico;100.10;900.00;F0.00;A4493240.04;T500.00
```

- (14 pontos) É necessário processar um arquivo contendo os clientes, suas respectivas contas e os ativos financeiros em seu nome. Os ativos financeiros podem ser ações, títulos públicos ou fundos imobiliários. Os ativos financeiros adotam os seguintes códigos de identificação: **A** — Ações; **T** — Títulos Públicos; **F** — Fundos Imobiliários.
  - (4 pontos) Defina a estrutura para criar um objeto do tipo **Conta**. Sua estrutura deve conter os campos **saldo** (double) e **limite** (double).
  - (4 pontos) Defina a estrutura para armazenar um **Ativo**. Sua estrutura deve conter os campos **tipo** (char) e **valor** (double).
  - (6 pontos) Defina a estrutura para armazenar um **Cliente**. Sua estrutura deve conter os campos **nome** (string), **ativos** (vetor do tipo Ativo com 3 posições), e **Conta** (ponteiro para valor do tipo Conta).

**banco.h**

```
1 #ifndef BANCO_H
2 #define BANCO_H
3 typedef struct Conta {
4
5
6 } Conta;
7 typedef struct Ativo {
8
9
10 } Ativo;
```

**banco.h**

```
11 typedef struct Cliente {
12
13
14
15 } Cliente;
16 Cliente processa_cliente(char *linha);
17 int processa_arquivo(char *nome_arquivo,
18                     Cliente **clientes);
19 int f(Cliente *clientes, int n, int k,
20      int i, Cliente *r);
21 #endif
```

- (30 pontos) Considere a seguinte estrutura **ResultadoProcValor**, que contém dois campos: **valor** (o valor processado) e **tamanho** (a quantidade de caracteres lidas).

**banco.c**

```
1 typedef struct ResultadoProcValor {
2     double valor; unsigned int tamanho;
3 } ResultadoProcValor;
```

**Atenção:** Nas questões a seguir, as únicas funções pré-definidas que você pode utilizar são **strlen**, **malloc** e **realloc**.

- (a) (15 pontos) Implemente uma função **processa\_valor**, que recebe como parâmetro uma string contendo um número real, *potencialmente* seguido de um ponto e vírgula (delimitador utilizado no arquivo CSV). Esta função deve processar o número real presente no início da string e retornar um valor do tipo **ResultadoProcValor**, onde o campo **valor** é o número que foi processado e o campo **tamanho** é a quantidade de caracteres lidos. Por exemplo, se a entrada for “23.15;10.23”: apenas o primeiro número será processado, então o valor será 23.15 e a quantidade de caracteres consumidos foi 5. Sua função deve ter a seguinte assinatura:

**ResultadoProcValor** processa\_valor(**char** \*linha);

- (b) (15 pontos) Implemente a função **processa\_cliente** declarada no cabeçalho **banco.h**. Esta função recebe como parâmetro uma string que corresponde a uma linha do arquivo CSV, e retorna um valor do tipo **Cliente**. Assuma que a string não contém `\n` no final. Se atente aos campos que devem ser alocados dinamicamente.
3. (26 pontos) A função **processa\_arquivo** percorre o arquivo CSV, lendo linha por linha e armazenando os clientes lidos em um vetor. Ela recebe como parâmetro o nome do arquivo (uma string), que deve ser aberto em modo *leitura*. O segundo parâmetro é um ponteiro para o vetor de clientes que será alocado e preenchido (note que o vetor já é um ponteiro, configurando assim um ponteiro para outro ponteiro; então, para acessar o vetor precisamos passar pelo primeiro ponteiro: **\*clientes**). A função retorna a quantidade de clientes processados. Você deve completá-la nos locais indicados. Vamos assumir que cada linha do arquivo tem, no máximo, 150 caracteres.

**banco.c**

```
1 #define TAMANHO_MAX_LINHA 150
2 int processa_arquivo(char *nome_arquivo, Cliente **clientes) {
3     // Abra o arquivo em modo leitura:
4     FILE *fp = _____;
5     if (fp == NULL) return -1;
6
7     int n = 0; int capacidade = 10;
8
9     // Aloque espaço para 10 clientes (utilize a variável capacidade):
10    *clientes = _____;
11    char linha[TAMANHO_MAX_LINHA + 1];
12
13    // A primeira linha do arquivo é o cabeçalho com o nome de cada coluna. Ela
14    // não será utilizada, mas precisamos passar por ela. Pule a primeira linha:
15    _____;
16
17    // Agora leia a segunda linha, que potencialmente é um cliente:
18    _____;
19
20    // Como saber se chegamos ao final do arquivo? Indique a condição:
21    while (_____) {
22        if (linha[strlen(linha) - 1] == '\n') linha[strlen(linha) - 1] = '\0';
23
24        Cliente c = processa_cliente(linha);
25        if (n >= capacidade) {
26            capacidade *= 2;
27            // Realoque o vetor de clientes de acordo com a nova capacidade:
28            *clientes = _____;
29        }
30
31        (*clientes)[n] = c; n++;
32
33        // Leia a próxima linha:
34        _____;
35    }
36    // Feche o arquivo:
37    _____;
38    return n;
39 }
```

4. (30 pontos) A função **f** a seguir foi implementada de forma recursiva. Ela assume que o vetor **r** passado por parâmetro já foi alocado e tem tamanho **n**, o mesmo tamanho que o vetor **clientes**.

**banco.c**

```

1  int f(Cliente *clientes, int n, int k, int i, Cliente *r) {
2      if (n == 0) {
3          return 0;
4      }
5
6      if (clientes[n - 1].conta->saldo >= k) {
7          r[i] = clientes[n - 1];
8          return 1 + f(clientes, n - 1, k, i + 1, r);
9      }
10
11     return f(clientes, n - 1, k, i, r);
12 }
```

Uma possível chamada inicial a esta função seria:

**int** m = f(clientes, n, 300.00, 0, r);

Descreva sucintamente o que a função faz. Indique qual o caso base e qual o caso recursivo. Dê exemplos da execução da função, mostrando o estado da pilha a cada chamada. Você não precisa mostrar todos os parâmetros e campos de estruturas detalhadamente, apenas os relevantes. Segue um exemplo do estado inicial da pilha, após a primeira chamada da função (assuma que a pilha cresce de cima para baixo):

