Introdução:

Um **arquivo binário** pode armazenar dados de qualquer maneira possível (por ser binário, há a possibilidade de organizar os *bytes* de qualquer combinação possível). Contudo, em geral, é estabelecido uma ordem pela qual os *bytes* são armazenados, de maneira a manter certa organização no arquivo, e de forma que os programas possam ler e armazenar esses bytes de uma forma padronizada. Imagine, por exemplo, que se não fosse feita essa padronização, não seria possível escrever programas que exibissem uma imagem no formato PNG na tela. Se não fosse padronizado o formato PNG, cada programa ia escrever e ler o arquivo binário de uma maneira diferente, ou seja, uma grande bagunça. Mas o PNG é um formato de arquivo binário, e foi PADRONIZADO que os *bytes* são organizados de uma maneira específica. Dessa forma, é possível que qualquer visualizador de imagens leia arquivos PNG e interpretem os bytes ali armazenados. Aqui na disciplina também vamos trabalhar com arquivos binários: não o PNG, mas arquivos binários que armazenam registros de dados.

Descrição:

Você deve implementar um programa que recebe como entrada o nome de um arquivo. Seu programa deve abrir o arquivo para escrita binária de forma a criar um novo arquivo (modo “wb”), realizar a leitura de registros a partir da entrada de texto e escrever os registros lidos para o arquivo binário. Os registros do arquivo binário em questão são organizados da seguinte maneira:

ID-DO-USUARIO (4 bytes, inteiro), NOME (tamanho variável, char), IDADE (4 bytes, inteiro)

Observe que os campos “ID-DO-USUARIO” e “IDADE” são inteiros (4 bytes). Então eles devem ser escritos a partir de uma variável do tipo “int” do C.

Observe que o campo “NOME” é uma *string do C* que pode variar de tamanho. Mas esse campo tem também o terminador de *string* do C (o ‘\0’), que vai dizer para você se o nome ali dentro ocupa 2 bytes, 30 bytes ou 49 bytes, por exemplo. Você deve escrever no disco todo o nome, incluindo o terminador de string (\0). Sendo assim, você vai escrever strlen(nome) + 1 bytes (contando com o \0).

Imagine, por exemplo, que o usuário digite como registro de entrada “1 Joao 30”. Você vai escrever então no arquivo binário na seguinte ordem: 4 bytes relativos ao inteiro do ID-DO-USUARIO (com valor = 1), 5 bytes relativos à *string* do NOME (com valor = “Joao\0”), 4 bytes relativos ao inteiro da IDADE (com valor = 30).

Entrada:

Primeiramente é passado como entrada o nome de um arquivo (que vai estar no formato binário padronizado acima) que você vai criar – um novo arquivo. Depois será passado um inteiro N que indica quantos registros serão salvos nesse arquivo. Depois, nas próximas N linhas, serão passados para você cada um dos registros no formato abaixo:

ID-DO-USUARIO NOME IDADE

Para facilitar, considere que nenhum desses campos nunca será nulo e o campo NOME não terá espaços (você pode ler normalmente usando scanf).

Saída:

IMPORTANTE: aqui a gente quer saber se você gerou o arquivo binário corretamente. Nesse caso, então, precisamos comparar o seu arquivo binário gerado com o nosso. Para isso vamos usar uma função chamada “binarioNaTela” (que vamos fornecer para você). Você não precisa entender essa função e nem como ela funciona. **Apenas entenda que**, ao final da execução de seu programa, antes de encerrar, você deve chamar essa função passando como argumento o nome do arquivo binário que você acabou de criar - que será comparado com o nosso arquivo binário. Essa função vai calcular um número em cima do seu arquivo e quanto mais próximo do número esperado, significa que mais correto seu arquivo binário está. A função segue abaixo (você pode só copiar ela no seu programa da forma como fornecemos e usar apenas no fim da execução):

```
void binarioNaTela(char *nomeArquivoBinario) { /* Você não precisa entender o código dessa função. */

	/* Use essa função para comparação no run.codes. Lembre-se de ter fechado (fclose) o arquivo anteriormente.
	*  Ela vai abrir de novo para leitura e depois fechar (você não vai perder pontos por isso se usar ela). */

	unsigned long i, cs;
	unsigned char *mb;
	size_t fl;
	FILE *fs;
	if(nomeArquivoBinario == NULL || !(fs = fopen(nomeArquivoBinario, "rb"))) {
		fprintf(stderr, "ERRO AO ESCREVER O BINARIO NA TELA (função binarioNaTela): não foi possível abrir o arquivo que me passou para leitura. Ele existe e você tá passando o nome certo? Você lembrou de fechar ele com fclose depois de usar?\n");
		return;
	}
	fseek(fs, 0, SEEK_END);
	fl = ftell(fs);
	fseek(fs, 0, SEEK_SET);
	mb = (unsigned char *) malloc(fl);
	fread(mb, 1, fl, fs);

	cs = 0;
	for(i = 0; i < fl; i++) {
		cs += (unsigned long) mb[i];
	}
	printf("%lf\n", (cs / (double) 100));
	free(mb);
	fclose(fs);
}
```

Observe então que a resposta que seu programa irá gerar será um número racional (você não precisa entender o significado desse número), e só fazemos isso por conta do sistema de correção do run.codes (não há como fazer correção de arquivo binário nativamente, então fazemos isso para poder usar o sistema deles).

Exemplo:

No começo não existe nenhum arquivo binário ainda (seu programa vai criar um).

A **entrada** do seu programa será:

```
registros.bin
6
1 Joao 30
2 Ana 15
3 Maria 28
4 MichaelScott 24
5 Jose 56
6 Leticia 10
```

A **saída** do seu programa deve ser um número (apenas execute a função “binarioNaTela” que a gente forneceu e isso já é feito automaticamente):

```
36.550000
```

Ao fim da execução, o arquivo binário **registros.bin** terá sido criado, e uma representação visual deste arquivo é:

```
id 1 – Joao – 30 anos
id 2 – Ana – 15 anos
id 3 – Maria – 28 anos
id 4 – MichaelScott – 24 anos
id 5 – Jose – 56 anos
id 6 – Leticia – 10 anos
```