# Laboratório 3

## Alocação dinâmica de memória, passagem de parâmetros, ponteiros para função e *templates* de função

## Objetivo

O objetivo deste exercício é colocar em prática o uso de alocação dinâmica de memória, passagem de parâmetros, ponteiros para função e *templates* de função através da implementação de um programa na linguagem de programação C++.

## Orientações gerais

Você deverá observar as seguintes observações gerais na implementação deste exercício:

1) Apesar da completa compatibilidade entre as linguagens de programação C e C++, seu código fonte não deverá conter recursos da linguagem C nem ser resultante de mescla entre as duas linguagens, o que é uma má prática de programação. Dessa forma, deverão ser utilizados estritamente recursos da linguagem C++.
2) Você deverá utilizar apenas um editor de texto simples (tais como o Gedit ou o Sublime) e o compilador em linha de comando, por meio do terminal do sistema operacional Linux.
3) Durante a compilação do seu código fonte, você deverá habilitar a exibição de mensagens de aviso (*warnings*), pois elas podem dar indícios de que o programa potencialmente possui problemas em sua implementação que podem se manifestar durante a sua execução.
4) Aplique boas práticas de programação. Codifique o programa de maneira legível (com indentação de código fonte, nomes consistentes, etc.) e documente-o adequadamente na forma de comentários. Anote ainda o código fonte para dar suporte à geração automática de documentação utilizando a ferramenta Doxygen (http://www.doxygen.org/). Consulte o documento extra disponibilizado na Turma Virtual do SIGAA com algumas instruções acerca do padrão de documentação e uso do Doxygen.
5) Busque desenvolver o seu programa com qualidade, garantindo que ele funcione de forma correta e eficiente. Pense também nas possíveis entradas que poderão ser utilizadas para testar apropriadamente o seu programa e trate adequadamente possíveis entradas consideradas inválidas.
6) Lembre-se de aplicar boas práticas de modularização, em termos da implementação de diferentes funções e separação entre arquivos cabeçalho (`.h`) e corpo (`.cpp`).

## Autoria e política de colaboração

O trabalho deverá ser feito **individualmente**. O trabalho em cooperação entre estudantes da turma é estimulado, sendo admissível a discussão de ideias e estratégias. Contudo, tal interação não deve ser entendida como permissão para utilização de (parte de) código fonte de colegas, o que pode caracterizar situação de plágio. Trabalhos copiados em todo ou em parte de outros colegas ou da Internet serão sumariamente rejeitados e receberão nota zero.

## Entrega

Você deverá submeter um único arquivo compactado no formato `.zip` contendo todos os códigos fonte resultantes da implementação das soluções às questões deste exercício, sem erros de compilação e devidamente testados e documentados na forma de comentários, **até o horário da aula do dia 4 de abril de 2017** através da opção *Tarefas* na Turma Virtual do SIGAA.

## Avaliação

O trabalho será avaliado sob os seguintes critérios: (i) utilização correta dos conteúdos vistos anteriormente e nas aulas presenciais da disciplina; (ii) a corretude da execução dos programas implementados, que devem apresentar saída em conformidade com a especificação e as entradas de dados fornecidas, e; (iii) a aplicação correta de boas práticas de programação, incluindo legibilidade, organização e documentação de código fonte. A presença de mensagens de aviso (*warnings*) ou de erros de compilação e/ou de execução, a modularização inapropriada e a ausência de documentação são faltas que serão penalizadas. Este trabalho contabilizará nota de até 1,0 ponto na 1ª Unidade da disciplina.

## Tarefa 1

Implemente uma função para manipulação de um vetor de inteiros com a seguinte assinatura:

```
int doVector (int* vetor, int tamanho, <tipo_operacao> operacao, int valor)
```

em que `<tipo_operacao>` deve ser implementado como uma **enumeração**. Nas linguagens de programação C e C++, uma enumeração (também chamado de tipo enumerável) é um tipo de dado definido pelo usuário que consiste em uma lista de constantes inteiras nomeadas, fazendo com que uma variável declarada com esse tipo poderá receber um dos valores definidos em tal enumeração. Com isso, a ideia por trás de uma enumeração é de possibilitar a criação de tipos de dados que se restringem a um conjunto determinado de valores. A enumeração `<tipo_operacao>` representará um conjunto contendo as seguintes operações:

| Operação | Descrição |
|---|---|
| `opMax` | Retorna o maior valor presente no vetor |
| `opMin` | Retorna o menor valor presente no vetor |
| `opSum` | Retorna a soma de todos os valores presentes no vetor |
| `opAvg` | Retorna o valor médio (parte inteira) de todos os valores presentes no vetor |
| `opHig` | Retorna a quantidade de elementos maiores que um determinado valor |
| `opLow` | Retorna a quantidade de elementos menores que um determinado valor |

Note que apenas as operações `opHig` e `opLow` requerem o quarto parâmetro, `valor`. Logo, o valor desse parâmetro para as funções que não o utilizam deverá ser, por *default*, zero. Além disso, a chamada da função referente à operação deverá ser realizada por meio de um *ponteiro de função*.

A título de exemplo, considere um vetor $v$ contendo cinco elementos $v$ = [12, 4, 60, 5, 23], de modo que o parâmetro `tamanho` tem valor 5. Chamadas e respectivas saídas para as operações descritas anteriormente seriam:

| Chamada | Resultado |
|---|---|
| `doVector(v, tamanho, opMax)` | 60 (o maior valor no vetor $v$) |
| `doVector(v, tamanho, opMin)` | 4 (o menor valor no vetor $v$) |
| `doVector(v, tamanho, opSum)` | 109 (a soma de todos os valores do vetor $v$) |
| `doVector(v, tamanho, opAvg)` | 21 (a parte inteira da média dos valores do vetor $v$, que é 21,8) |
| `doVector(v, tamanho, opHig, 25)` | 1 (quantidade de elementos do vetor $v$ cujo valor é maior que 25) |
| `doVector(v, tamanho, opLow, 15)` | 3 (quantidade de elementos do vetor $v$ cujo valor é menor que 15) |

## Tarefa 2

Utilizando **ponteiros genéricos**, **sobrecarga de funções** e/ou ***templates* de função**, apresente uma versão genérica da função `doVector` implementada na Tarefa 1 a fim de permitir que as operações anteriormente descritas possam também ser executadas sobre números reais (tipos `float` ou `double`).

## Tarefa 3

Utilizando ***templates* de função**, apresente uma versão especializada das funções genéricas implementadas na Tarefa 2 a fim de permitir que as operações anteriormente descritas possam também ser executadas sobre *strings*. Neste caso específico, para um vetor de *strings*, essas funções deverão se comportar da seguinte forma:

| Operação | Descrição |
|---|---|
| `opMax` | Retorna a *string* de maior tamanho no vetor |
| `opMin` | Retorna a *string* de menor tamanho no vetor |
| `opSum` | Retorna a soma dos tamanhos de todas as *strings* contidas no vetor |
| `opAvg` | Retorna a média dos tamanhos de todas as *strings* contidas no vetor |
| `opHig` | Retorna o número de *strings* cujo tamanho é maior que um determinado valor |
| `opLow` | Retorna o número de *strings* cujo tamanho é menor que um determinado valor |

## Tarefa 4

Desenvolva um programa para testar a implementação final (resultado das Tarefas 1, 2 e 3). O programa deverá primeiramente solicitar ao usuário o tipo de dados a serem armazenados no vetor (`int`, `float`, `double` ou `string`) e o tamanho do vetor. O vetor deverá ser **alocado dinamicamente** em memória e preenchido com **valores randômicos** variando entre 1 e a quantidade de elementos informada pelo usuário para o caso de valores de tipos numéricos, e preenchido com *strings* informadas pelo usuário para o caso de valores do tipo *string*. Feito isso, o programa imprime os valores contidos no vetor e solicita ao usuário uma dentre as seis possíveis opções de operações, numeradas de 1 a 6, sendo encerrado se o usuário informar zero como opção ou exibindo mensagem de erro caso seja escolhida uma opção inválida. Por fim, o programa exibe o resultado da execução da operação selecionada pelo usuário. Para auxilia-lo na execução desta tarefa, você poderá utilizar como ponto de partida o trecho de código a seguir (também disponível para *download* através da Turma Virtual do SIGAA), que **ainda necessita ser completado**.

```
1:  int main() {
2:     string tipo_dados;          // Tipo de dados (int, float, double, string)
3:     cout << "Tipo de dados: ";
4:     cin >> tipo_dados;
5:
6:     int tamanho;
7:     cout << "Informe a quantidade de elementos: ";
8:     cin >> tamanho;
9:
10:    // Possiveis vetores a serem alocados dinamicamente conforme o tipo de dados
11:    int* vint = NULL;           // Vetor de inteiros
12:    float* vfloat = NULL;       // Vetor de decimais (float)
13:    double* vdouble = NULL;     // Vetor de decimais (double)
14:    string* vstring = NULL;     // Vetor de strings
15:
```

*Continua na próxima página*

```
16:     // Alocacao, preenchumento e impressao do vetor de acordo com o tipo de dados
17:     if (tipo_dados == "int") {
18:         // Alocar dinamicamente o vetor de inteiros
19:         // Preencher o vetor chamando funcao generica
20:         // Imprimir o vetor chamando funcao generica
21:     }
22:
23:     // Executar as mesmas instrucoes para os demais tipos de dados
24:
25:     // Apresentar opcoes de operacoes ao usuario
26:
27:     int operacao = 0;     // Opcao de operacao a ser escolhida pelo usuario
28:     cin >> operacao;
29:     if (operacao != 0) {
30:         if (tipo_dados == "int") {
31:             doVector(vint, tamanho, tipo_operacao(op), 0);
32:         } else if (tipo_dados == "float") {
33:             doVector(vfloat, tamahho, tipo_operacao(op), 0);
34:         } else if (tipo_dados == "double") {
35:             doVector(vdouble, tamanho, tipo_operacao(op), 0);
36:         } else {
37:             doVector(vstring, tamanho, tipo_operacao(op), 0);
38:         }
39:     }
40:
41:     cout << "Programa encerrado." << endl;
42:     return 0;
43: }
```

Um exemplo de execução do programa, que gera um vetor contendo seis elementos inteiros e realiza as operações mediante sucessivas entradas do usuário, seria:

```
$ ./doVector
Tipo de dados: int
Informe a quantidade de elementos: 6
Vetor: [4, 1, 6, 2, 3, 5]

Operacoes:
(1) Maior valor
(2) Menor valor
(3) Soma dos elementos
(4) Media dos elementos
(5) Elementos maiores que um valor
(6) Elementos menores que um valor
(0) Sair
```

*Continua na próxima página*

```
Digite sua opcao: 1
Maior valor no vetor: 6

Digite sua opcao: 2
Menor valor no vetor: 1

Digite sua opcao: 3
Soma dos elementos do vetor: 21

Digite sua opcao: 4
Media dos elementos do vetor: 3

Digite sua opcao: 5
Informe o valor limite: 2
Quantidade de elementos maiores que 2: 4

Digite sua opcao: 6
Informe o valor base: 4
Quantidade de elementos maiores que 4: 3

Digite sua opcao: 0
Programa encerrado.
```