Ponteiros

- **Ponteiros** são um tipo de variável que aponta para outra, ou seja, um ponteiro guarda o endereço de memória de uma variável. No exemplo abaixo, observe que: **(i)** a variável `p` é do tipo ponteiro (não é do tipo inteiro); **(ii)** o símbolo **&** indica o endereço de memória da variável, e; **(iii)** para acessar o valor de uma variável apontada por um ponteiro, usamos asterisco ***** antes da variável ponteiro.

```
#include<iostream>
using namespace std;
main(){
	int i=20;
	int *p;
	p=&i;
	cout << "Valor da variável: " << *p << endl;
}
```

Malloc e Free

- Utilizando ponteiros, podemos declarar um vetor e seu tipo e somente depois de ler uma informação do usuário, alocar o espaço do vetor na memória. Para isso, utilizamos o comando `malloc`. Para liberar o espaço ocupado em memória que havia sido alocado usando `malloc`, utilizamos o comando `free`. **Exemplo**:

```
#include<iostream>
#include<stdlib.h>  // malloc, free
using namespace std;

main(){
	int *ra;
	float *notas;
	int n;
	int i;
	cout << "Entre com o total de alunos: ";
	cin >> n;
	ra = (int *) malloc(n * sizeof(int)); // aloca n posições
	notas = (float *) malloc(n * sizeof(float));//aloca n posições
	for (int j=0; j<n; j++){
		cout << endl << "Entre com o RA: ";
		cin >> ra[j];
		cout << endl << "Entre com a nota: ";
		cin >> notas[j];
	}
	for (int j=0; j<n; j++){
```

```
                        cout << "O aluno de RA "<< ra[j] << " tem nota " <<
notas[j] << endl;
                }
                free(ra); // libera memória ocupada pelo vetor ra
                free(notas); // libera memória ocupada pelo vetor notas
        }
```

Aritmética de Ponteiros

- Um vetor pode ser acessado como um ponteiro. Assim, a primeira posição de um vetor é a mesma coisa que um ponteiro que aponta para esse vetor. Isso porque um ponteiro só pode apontar para a primeira posição de um vetor. Usando a aritmética de ponteiros podemos adicionar (ou subtrair) `x` unidades de um ponteiro para que o ponteiro se mova para mais (ou menos) `x` unidades, onde a unidade é o tamanho do tipo de dados para o qual ele aponta. Isso significa que, para um vetor `V` e um ponteiro `p` que aponta para `V`, temos

```
V[0]==*(p);
V[1]==*(p+1);
V[2]==*(p+2);
   :     :
```

- **Exemplo:**

```
#include<iostream>
using namespace std;
main(){
        int V[50];
        int *pont;
        pont=V;// não precisa "&" pq vetores já são representados como
ponteiro para 1a posição
        for (int i=0; i<15; i++){
                V[i]=i+10;
        }
        cout << "A 1a nota é: " << V[0] << endl;       // Imprime 10
        cout << "A 1a nota é: " << *pont << endl;      // Imprime 10
        cout << "A 10a nota é: " << V[9] << endl;      // Imprime 19
        cout << "A 10a nota é: " << *(pont+9) << endl; // Imprime 19
}
```

Métodos

- **Variáveis locais e globais**: variáveis locais são acessíveis somente dentro do método no qual são declaradas. Variáveis globais são acessíveis em qualquer parte do programa;
- A função `main()`: é a função principal de um programa em C/C++. Todo programa deve ter a função `main()`, caso contrário o compilador reclama e não gera o executável. Um programa começa executando a função `main()` e termina quando esta função termina.

- **Dois tipos de métodos:**

  (1) Com retorno: realizam processamento e devolvem uma informação. **Exemplo:**

```
int fatorial(int n){
	int fat=1;
	for(int i=2; i<=n; i++)
		fat=fat*i;
	return fat;
}
```

  (2) Sem retorno: realizam tarefas e operações sem devolver uma informação. **Exemplo:**

```
void fatorial(int n){
	int fat=1;
	for(int i=2; i<=n; i++)
		fat=fat*i;
	cout << "(Sem retorno) - Fatorial de " << n << " é " << fat << endl;
}
```

- **Duas formas de passar parâmetros:**

  (1) Por valor: os valores das variáveis de entrada de uma função são copiados para ela. As duas funções acima passam o parâmetro `n` por valor;

  (2) Por referência: o endereço na memória das variáveis é passado como parâmetro para a função que pode modificar o valor da variável. **Exemplo**:

```
#include<iostream>
using namespace std;
// variáveis globais
int exemplo_var_global=0;

void troca(float *a, float *b) {
	float temp;
	temp = *a;
```

```
        *a = *b;
        *b = temp;
    }
    main(){
        float a=10.5, b=17.1;
        cout << "Antes: a=" << a << " b=" << b << endl;
        troca(&a,&b);
        cout << "Depois: a=" << a << " b=" << b << endl;
    }
```

Protótipo de Função

- É uma técnica muito utilizada em programas C e C++, na qual as funções são declaradas antes de serem definidas. **Exemplo:**

```
#include<iostream>
using namespace std;

//criando o protótipo de 4 funções
int somar (int x, int y);
int subtrair (int x, int y);

main(){
    int n, m, Som, Sub;
    cout << "Digite o 1o valor: ";
    cin >> n;
    cout << "Digite o 2o valor: ";
    cin >> m;
    //chamando todas as funções
    Som = somar (n, m);
    Sub = subtrair (n, m);

    cout << n << " + " << m << " = " << Som << endl;
    cout << n << " - " << m << " = " << Sub << endl;
}

//As funções
int somar (int x, int y){
    return x+y;
}

int subtrair (int x, int y){
    return x-y;
}
```

Utilizando bibliotecas em C

- Algumas funções úteis de biblioteca:

Nome da Função	Utilidade	Biblioteca
abs(n)	Retorna o valor absoluto de n	math.h
ceil(n)	Retorna a função teto de n	math.h
cos(n)	Calcula coseno de n	math.h
floor(n)	Retorna a função piso de n	math.h
pow(n,m)	Calcula n elevado a m	math.h
random(n)	Retorna um número aleatório entre 0 e n-1	stdlib.h
sin(n)	Calcula seno de n	math.h
sqrt(n)	Calcula raiz quadrada de n	math.h
tan(n)	Calcula tangente de n	math.h
tolower(n)	Converte o caractere n para minúsculo	ctype.h
toupper(n)	Converte o caractere n para maiúsculo	ctype.h

- **Exercício.** Para uma lista estática (vetor), implemente os métodos das três operações básicas: (a) inserção; (b) remoção, e; (c) busca. Observações:
  - As operações de inserção e remoção devem ter dois métodos diferentes cada, um para listas ordenadas e outro para listas não ordenadas;
  - Para a operação de busca devem ser implementadas duas funções recursivas: (1) uma busca sequencial (para lista não ordenada) e (2) uma busca binária (para lista ordenada).