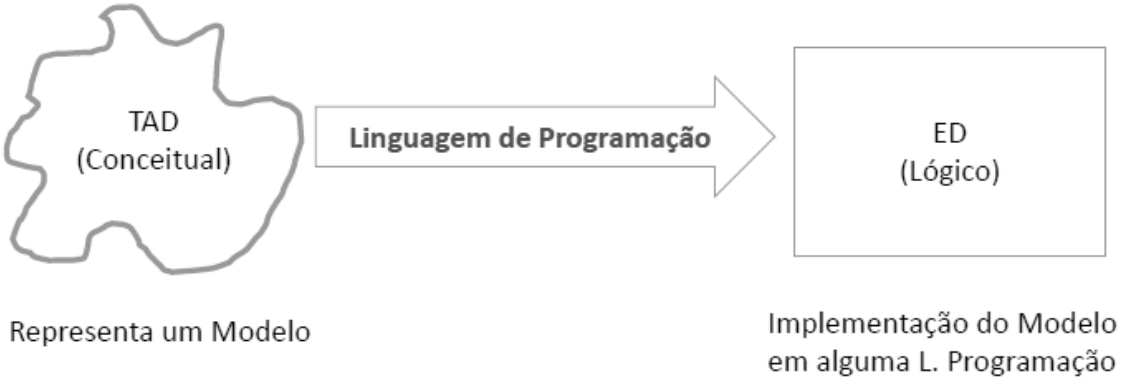
Fatec Santos - Estrutura de Dados
ANTHONY CLINT PASTORELLO R.A.: 0050482011005
Prof. Chiara

1. **O que podemos entender por um tipo abstrato de dados? E uma estrutura de dados?**

   Tipo abstrato de dados (TAD) é a representação de um conjunto de dados e um conjunto de procedimentos (funções) que podem ser aplicadas sobre esses dados.

   Por outro lado, uma estrutura de dados (ED) é uma implementação de um tipo abstrato de dados, representa as relações lógicas entre esses dados.



2. **O que são listas lineares?**

Lista linear é uma estrutura de dados na qual elementos de um mesmo tipo de dado estão organizados de maneira sequencial. Como, por exemplo, a sequência de elementos x1 , x2 , x3 ,..., xn , tal que:

— x1 é o primeiro elemento da Lista;
— xn é o último elemento da Lista;
— Para todo i, j, se i <j então xi é o antecessor de xj;
— Para todo i, j, se i >j então xi é o sucessor de xj;
— Se n=0, então a lista está vazia;
— n é a quantidade de elementos da lista.

3. **O que são listas lineares restritas? Cite 2 tipos e respectivas definições.**

Listas lineares restritas são listas cujas operações de dados ocorrem apenas por um “lugar” (topo ou extremidades). Dois exemplos de lista restrita são:

Uma pilha pelo fato das operações de acesso (top), inserção (push) e remoção (pop) serem realizadas somente em uma das extremidades chamada topo.

Uma fila pelo fato das operações de acesso e remoção serem realizadas em uma das extremidades chamada início e a operação de inserção ser realizada na outra extremidade chama fim.

4. **Considere os protótipos, definidos abaixo, para uma estrutura de dados do tipo fila, implementada em alocação sequencial com representação circular.**

void insere(int x): //insere um novo elemento na fila

int remove(): //remove um elemento da fila, retornando este elemento removido

int primeiro(): //retorna o valor do primeiro elemento da fila.

Mostre a situação de uma fila, inicialmente vazia após cada uma das seguintes operações:

1. insere(5)	2. insere(8)	3. insere(2)
4. insere(primeiro())	5. remove()	6. insere(remove())
7. remove()	8. insere(1)	9. remove()
10. insere(remove())	11. insere(primeiro())	12. insere(remove())
13. insere(3)	14. insere(remove())	15. insere(3);

1. insere(5)
Fila: 5

2. insere(8)
Fila: 5 8

3. insere(2)
Fila: 5 8 2

4. insere(primeiro())
Fila: 5 8 2 5

5. remove()
Fila: 8 2 5

6. insere(remove())
Fila: 2 5 8

7. remove()
Fila: 5 8

8. insere(1)
Fila: 5 8 1

9. remove()
Fila: 8 1

10. insere(remove())
Fila: 1 8

11. insere(primeiro())
Fila: 1 8 1

12. insere(remove())
Fila: 8 1 1

13. insere(3)
Fila: 8 1 1 3

14. insere(remove())
Fila: 1 1 3 8

15. insere(3)
Fila: 1 1 3 8 3

5. **Considere os protótipos, definidos abaixo, para uma estrutura de dados do tipo Pilha, implementada em alocação sequencial:**

a) void empilha(int x): - empilha um novo elemento

b) int desempilha(): - desempilha um elemento e retornando o elemento desempilhado

c) int topo(): - retorna o valor do elemento do topo da pilha

Mostre a situação de uma pilha , inicialmente vazia, após a execução de cada umas das operações:

1. empilha(1)	2. empilha(5)	3. desempilha()
4. empilha(5)	5. empilha(9)	6. empilha(7)
7. empilha(topo())	8. desempilha()	9. empilha(desempilha())
10. empilha(4)	11. empilha(3)	12. empilha(8)

1. empilha(1)
Pilha: 1

2. empilha(5)
Pilha: 5 1

3. desempilha()
Pilha: 1

4. empilha(5)
Pilha: 5 1

5. empilha(9)
Pilha: 9 5 1

6. empilha(7)
Pilha: 7 9 5 1

7. empilha(topo())
Pilha: 7 7 9 5 1

8. desempilha()
Pilha: 7 9 5 1

9. empilha(desempilha())
Pilha: 7 9 5 1

10. empilha(4)
Pilha: 4 7 9 5 1

11. empilha(3)
Pilha: 3 4 7 9 5 1

12. empilha(8)
Pilha: 8 3 4 7 9 5 1

6. **Utilizando as implementações de uma estrutura de dados pilha, elabore um programa que converta um número decimal em número binário.**

```
#include<iostream>
#include "PILHA_P1.h"
#include<locale.h>
#include<stdio.h>
#include<stdlib.h>
#include<Windows.h>

using namespace std;

HANDLE color = GetStdHandle(STD_OUTPUT_HANDLE);

Pilha<int> p(16);

void bina(int deci)
{
    if(deci == 0)
    {
        p.empilha(0);
    }
    else
    {
        while (deci != 0)
        {
            if (deci % 2 == 0)
            {
                p.empilha(0);
            }
            else
            {
                p.empilha(1);
            }

            deci /= 2;
        }
    }
}

void exibe(int deci)
{
    cout << "\n\n\t\tNúmero decimal: " << deci << endl;
    cout << "\t\tNúmero em binário: ";

    for (int i = p.getTopo(); i >= 0; i--)
    {
        cout << p.getVal(i) << " ";
    }
    cout << "\n\n";
}
```

```
int main()
{
    int deci;
    char resp;

    setlocale(LC_ALL, "portuguese");
    do
    {
        do
        {
            SetConsoleTextAttribute(color, 15); // letra branca

            cout << "\n\n\t\tCONVERSÃO DE DECIMAL PARA BINÁRIO\n" << endl;

            cout << "\n\t\tINFORME UM NÚMERO DECIMAL: ";
            cin >> deci;

            if (deci < 0)
            {
                SetConsoleTextAttribute(color, 12); // letra vermelha

                cout << "\n\n\t\t VALOR INVÁLIDO!!!" << endl;

                Sleep(2000);

                system("cls");
            }

        } while (deci < 0);

        bina(deci);
        exibe(deci);

        cout << "\n\n\t\tDESEJA INFORMAR OUTRO NÚMERO?(S/N): ";
        cin >> resp;


        if(resp == 's' || resp == 'S')
        {
            while(!p.emptypilha())
            {
                p.desempilha();
            }
            system("cls");
        }
        else
        {

            system("cls");
            cout << "\n\n\t\t>>> PROGRAMA FINALIZADO <<<\n\n\n" << endl;
        }

    } while (resp == 's' || resp == 'S');

    return 0;
}
```

7. **Considerando uma pilha e uma fila, de números inteiros, construa uma aplicação que:**
   a) define uma pilha e uma fila de tamanho 20.
   b) carrega a fila com números aleatórios compreendidos entre 35 e 78
   b) exibe a fila
   c) transfere todos os elementos da fila para a pilha
   d) exibe a pilha

```
#include<iostream>
#include "PILHA_P1.h"
#include "FILA_P1.h"
#include<locale.h>
#include<stdio.h>
#include<stdlib.h>
#include<Windows.h>
#include<time.h>

using namespace std;

HANDLE color = GetStdHandle(STD_OUTPUT_HANDLE);

Pilha<int> p(20); // a)
Fila<int> f(20);  //a)

int main()
{
   char resp;
   setlocale(LC_ALL, "portuguese");

   do
   {
      system("cls");

      srand(time(NULL));

      for (int i = 0; i < 20; i++)
      {
         f.put(35 + rand() % 44); // b)
      }

      SetConsoleTextAttribute(color, 15); // letra branca

      cout << "\n\n\t\tFILA: ";

      SetConsoleTextAttribute(color, 14); // letra amarela

      if (f.getBegin() > f.getEnd()) // c)
      {
         for (int i = f.getBegin(); i < f.getSize(); i++)
         {
            cout << f.getVal(i) << " ";
         }
         for (int i = 0; i <= f.getEnd(); i++)
         {
            cout << f.getVal(i) << " ";
         }
      }
      else
```

```
    {
        for (int i = f.getBegin(); i <= f.getEnd(); i++)
        {
            cout << f.getVal(i) << " ";
        }
    }

    cout << "\n\n";

    for (int i = 0; i < 20; i++)
    {
        p.empilha(f.getVal(i)); // d)
    }

    SetConsoleTextAttribute(color, 15); // letra branca

    cout << "\n\n\t\tPILHA: ";

    SetConsoleTextAttribute(color, 14); // letra amarela

    for (int i = p.getTopo(); i >= 0; i--) // e)
    {
        cout << p.getVal(i) << " ";
    }
    cout << "\n\n";

    SetConsoleTextAttribute(color, 15); // letra branca

    cout << "\n\n\t\tDESEJA REPETIR?(S/N): ";
    cin >> resp;

    if (resp == 's' || resp == 'S')
    {
        while (!p.emptypilha())
        {
            p.desempilha();
        }
        system("cls");
    }
    else
    {
        system("cls");
        cout << "\n\n\n\n\n\t\t\t>>>>> PROGRAMA FINALIZADO <<<<<\n\n\n\n\n\n" << endl;
    }

} while (resp == 's' || resp == 'S');

return 0;
      }
```

8. **Um pangrama é uma frase que contém pelo menos uma vez cada uma das 26 letras do novo alfabeto Português. Um exemplo de pangrama é: “UM PEQUENO JABUTI XERETA CHAMADO KYA VIU DEZ CEGONHAS FELIZES E GRITOU IWUP, IWUP!”**
Construa uma aplicação que recebe uma frase e verifica se ela é pangrama (utilize os conceitos de listas e strings da linguagem C++).
Frases para teste:
   - jackdawf loves my big quartz sphinx
   - abcdefghijklmnopqrstuvwxyz
   - the quick brown fox jumps over a lazy dog
   - jackdaws loves my big sphinx of quartz
   - hello world
   - esta frase es muy larga y contiene todas las letras abc def ghij klmnopqr stu vw x y zzzzz
   - supercalifragilistico espialidoso
   - alfa beta gamma delta epsilon iotta kappa lambda

```
#include<iostream>
#include "PILHA_LIGADA.h"
#include<locale.h>
#include<stdio.h>
#include<stdlib.h>
#include<Windows.h>
#include<string>
#include<sstream>

using namespace std;

HANDLE color = GetStdHandle(STD_OUTPUT_HANDLE);

int main()
{
   setlocale(LC_ALL, "Portuguese");

   Pilha_Lig<char> p;
   string frase;
   char c;

   do
   {
      system("cls");

      SetConsoleTextAttribute(color, 15); // letra branca

      cout << "\n\n\t\t DETECTOR DE PANGRAMA" << endl;

      cout << "\n\n\t\t INSIRA A FRASE A SER VERIFICADA: ";
      getline(cin, frase);

      unsigned i = 0;

      int M = 65; // A
      int m = 97; // a

      for (i; i < frase.length(); i++)
      {
         c = frase.at(i);

         if (M != 91) // [ --> primeiro simbolo depois do Z
```

```
            {
                if (c == M || c == m)
                {
                    p.empilha(c);

                    M++;
                    m++;

                    i = -1;
                }
            }
            else
            {
                i = frase.length();
            }
        }

        if (p.elementodotopo() == 'z' || p.elementodotopo() == 'Z')
        {
            SetConsoleTextAttribute(color, 10); // letra verde

            cout << "\n\n\t\t A FRASE É UM PANGRAMA!!!" << endl;
        }
        else
        {
            SetConsoleTextAttribute(color, 12); // letra vermelha

            cout << "\n\n\t\t A FRASE NÃO É UM PANGRAMA!!!" << endl;
        }

        SetConsoleTextAttribute(color, 15); // letra branca

        cout << "\n\n\t\t DESEJA REPETIR?(S/N):  ";
        cin >> c;
        cin.ignore();

        if (c != 's' && c != 'S')
        {
            system("cls");

            SetConsoleTextAttribute(color, 10); // letra verde

            cout << "\n\n\n\n\n\t\t\t>>>>> PROGRAMA FINALIZADO <<<<<\n\n" << endl;

            SetConsoleTextAttribute(color, 15); // letra branca
        }
        else
        {
            while (!p.pilhavazia())
            {
                p.desempilha();
            }
        }

    } while (c == 's' || c == 'S');

    return 0;
}
```

ANEXO 1

```
#ifndef PILHA_P1
#define PILHA_P1
template<typename Tipo>

class Pilha
{
private:
	Tipo* v;
	unsigned tamanho;
	int topo;

public:
	Pilha(unsigned tam)
	{
		tamanho = tam;
		v = new Tipo[tamanho];
		topo = -1;
	}

	~Pilha()
	{
		delete[]v;
	}

	void empilha(Tipo x)
	{
		topo++;
		v[topo] = x;
	}

	Tipo desempilha()
	{
		Tipo temp = v[topo];
		topo--;
		return temp;
	}

	Tipo elementodotopo()
	{
		return v[topo];
	}

	bool fullpilha()
	{
		return topo == (tamanho - 1);
	}

	bool emptypilha()
	{
		return topo == -1;
	}

	unsigned getTopo()
	{
		return topo;
	}
```

```
        unsigned getTamanho()
        {
                return tamanho;
        }

        Tipo getVal(unsigned posicao)
        {
                return v[posicao];
        }
};
    #endif // PILHA_P1
```

ANEXO 2

```
#ifndef FILA_P1
#define FILA_P1
template<typename Tipo>

class Fila
{
private:
	int begin, end;
	unsigned qtd, size;
	Tipo* v;

public:
	Fila(unsigned tam)
	{
		end = -1;
		begin = 0;
		qtd = 0;
		size = tam;
		v = new Tipo[size];
	}

	~Fila()
	{
		delete[]v;
	}

	void put(Tipo x) // função que insere na lista
	{
		end++;
		if (end == size)
		{
			end = 0;
		}
		v[end] = x;
		qtd++;
	}

	Tipo takeout() // função que retira da lista
	{
		Tipo temp = v[begin];
		begin++;
		if (begin == size)
		{
			begin = 0;
		}

		qtd--;

		return temp;
	}

	Tipo first() // função primeiro da fila
	{
		return v[begin];
	}
```

```
        bool fullist() // função lista cheia
        {
                return qtd == size;
        }

        bool emptylist() // função lista vazia
        {
                return qtd;
        }

        // as funções get servem pra dar acesso às variáveis que estão encapsuladas

        int getEnd()
        {
                return end;
        }

        int getBegin()
        {
                return begin;
        }

        unsigned getQtd()
        {
                return qtd;
        }

        unsigned getSize()
        {
                return size;
        }

        Tipo getVal(unsigned pos)
        {
                return v[pos];
        }
};
#endif // FILA_P1
```

ANEXO 3

```
#ifndef PILHA_LIGADA
#define PILHA_LIGADA
template <typename Tipo>
struct Node
{
	Tipo info;
	Node<Tipo>* prox;
};
template <typename Tipo>
class Pilha_Lig
{
private:
	Node<Tipo>* topo;

public:
	Pilha_Lig()
	{
		topo = NULL;
	}

	bool empilha(Tipo x)
	{
		bool v = true;
		Node<Tipo>* aux = new Node<Tipo>; // passo 1 - alocação
		aux->info = x; // passo 2 - guardar
		aux->prox = topo; // passo 3 - aponta pra NULL
		topo = aux; // passo 4
		return v;
	}

	Tipo desempilha()
	{
		Tipo temp = topo->info; // passo 0 - guardando o elemento
		Node<Tipo>* aux = topo; // passo 1
		topo = topo->prox; // passo 2
		delete aux; // passo 3
		return temp;
	}

	Tipo elementodotopo()
	{
		return topo->info;
	}

	bool pilhavazia()
	{
		return topo == NULL;
	}

	Node<Tipo>* getopo()
	{
		return topo;
	}

};

#endif // PILHA_LIGADA
```