INSTITUTO DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PARANÁ – IFPR

Campus Foz do Iguaçu

Disciplina – Algoritmos e Linguagem de Programação

Professor Jefferson Chaves

jefferson.chaves@ifpr.edu.br

Aluno: Vinicius de Oliveira Jimenez

jimenezvini@gmail.com

Aluno: Matheus Laurentino Barbosa

matheus.laurentino.ifpr@gmail.com

Aluno: Dyogo Romagna Bendo

dyogoromagnabendo@gmail.com

ESTUDO DIRIGIDO 2 – RESPOSTAS

Teórica:

1) Podemos usar vetor, array ou arranjos.

2) Em relação às variáveis normais, as Listas possibilitam que uma variável possa guardar mais de uma informação seja guardada. Isso se decorre, pois, as Listas possibilitam que haja mais de uma posição a ser ocupada por valores. De tal modo, a cada posição é possível guardar uma informação e podemos, de tal forma, não perder dados que possam ser essenciais para o programa.

3) a. Toda lista possui um **índice** número que se inicia em 0;
b. Uma lista nativa em cpp tem um tamanho **determinado**;

c. Uma lista usando a classe Vector tem como principal característica ter tamanho **dinâmico**;

4) Significa percorrer os elementos de uma Lista.

5) As formas mais comuns usadas para iterar uma Lista são usando as seguintes estruturas:

```
//1º maneira de iterar uma lista//

for (int i = 0; i < tamanho_lista; i ++){
    cout << lista[i]; // de forma que i representa as posições da Lista
}
```

```
//2º maneira de iterar uma lista//

for (int (deve ser do mesmo tipo da lista) listar : lista)
{
    cout << listar;
}
```

```
//3º maneira de iterar uma lista//


int i = 0;
while (i < tamanho_lista)
{
    cout << lista[i];
}
```

É possível copiar e colar o mesmo código apenas substituindo a posição, porém se demonstra pouco eficiente tal método, especialmente para grandes quantias de elementos.

6) Funções são códigos feitos para auxiliar o código principal. As funções realizam algum papel, e sempre que esse papel for necessário dentro do código, elas são chamadas.

7) As funções agilizam a construção de um programa, além de serem mais práticas para serem feitas. Ao mesmo tempo, se torna mais fácil para corrigir possíveis erros do código.

8) Declarar uma função é dizer da sua existência, resumidamente. Ao declarar, o compilador passa a saber que existe tal função e que ela pode ser usada

posteriormente. Na declaração, é necessário informar seus parâmetros (se possuir) assim como seu tipo, caso retorne algum valor ou se for void (vazio). Chamar uma função consiste em executar o bloco de códigos da função, informando os argumentos que tal função necessita para funcionar, se precisar. Esse fluxo de funcionamento é necessário pois senão o compilador ficaria perdido, pois se chamássemos a função antes de declará-la ele não saberia identificar que é uma função. Ao mesmo tempo, chamar a função facilita sabermos que é aquela função que foi declarada anteriormente e faz algum bloco de instrução que está sendo utilizado.

9) Os parâmetros são valores que serão dados pelos argumentos da função durante a execução do código principal. Esses valores serão úteis para o funcionamento do código da função.

10) O retorno é o valor que a função passa a ter. Por isso, funções com retorno precisam ter seu tipo indicado. No momento que é colocado no código, a função imediatamente para e o código principal retoma.

11) Poderia, pois existiam várias questões que faziam alguma coisa semelhante e era necessário copiar e colar um código, como nas somas. Se houvesse uma função de somar, poderíamos realizar a tarefa de somar e parcialmente a tarefa de realizar a média, agilizando a programação.

Algoritmos de Repetição:

1)

```
for (int i = 3; i < 12; i = i + 3) {
    cout << i;
}
```

resposta: 369

2)

```
int foo = 1;

while (foo <= 5) {
    cout << foo;
    foo++;
}
```

resposta: 12345

Listas:

1)

```
int lista [5] = {1, 2, 3, 4, 5};

for (int j = 0; j < 5; ++j) {
    cout << lista [j];
```

resposta: 12345

2)

```
int lista [5] = {7, 14, 99, 3 ,5};
int x = lista [0];

for (int k = 1; k < 5; k++) {
    if (lista[k] > x) {
        x = lista[k];
    }
}
cout << x;
```

resposta: 99

3)

```
vector <string> cidades = {“Florianopolis”, “Curitiba” Medianeira”,
“Joinville”};

sort(cidades.begin(), cidades.end());

for (int i = 0; i < cidades.size(); i++) {
    cout << cidades[i];
}
```

resposta: CuritibaMedianeiraFlorianopolesJoinville