Aula 5

Linguagem de Programação

Prof. Sandro de Araújo

Conversa Inicial

Recursividade e macros

- Esta aula apresenta a seguinte estrutura de conteúdo:

1. Recursividade
2. Recursão *versus* iteração
3. Função recursiva
4. Função recursiva com vetor
5. Funções macro – diretiva #define

- O objetivo desta aula é apresentar os principais conceitos e aplicações de recursividade, iteração, função recursiva, função recursiva com vetor e função macro com a diretiva #define, bem como a representação destes em linguagem C para criação de algoritmos

Recursividade

- A recursividade ou recursão se dá quando uma função chama a si mesma para resolver um problema
- E como funciona a recursividade?
  - De um modo geral, a recursividade é considerada como um processo repetitivo de uma rotina (procedimento ou função) que faz uma chamada para ela mesma

- **Recursão direta – É uma rotina composta por um conjunto de instruções, das quais uma delas faz a chamada para a rotina. A rotina X chama a própria rotina X**
- **Recursão indireta – É uma rotina que contém uma chamada a outra rotina, a qual tem uma chamada a outra rotina, e assim sucessivamente. A rotina X chama uma rotina Y que, por sua vez, chama X**

Exemplos de recursividade em nossa vida:

- **Caracóis**
- **Girassóis**
- **As folhas de algumas árvores**
- **Dois espelhos quando apontados um para o outro**
- **Entre outros**

Recursão *versus* iteração

- **Quando devemos usar a recursão ou iteração em nosso algoritmo?**
- **A regra é:**
  - **É possível resolver o problema com iteração?**
    - **Se a resposta for *sim*, então é possível resolver o mesmo problema com recursão**

- **As duas formas precisam de uma condição para terminar o ciclo repetitivo – um teste de terminação**

- **A iteração se encerra quando a condição de teste falha, já a recursão se encerra quando se alcança o caso trivial**

- A iteração e a recursão podem ingressar em LOOP infinito
  - Na iteração, se o teste jamais se tornar falso, o laço vai se repetir eternamente
  - Na recursão, se o problema não for reduzido de forma que se converta para o caso trivial, o laço vai se repetir até sobrecarregar a memória

- Analise o tempo computacional que será gasto com uma função recursiva e veja se vale ou não a pena, se o gasto for muito alto, implemente a mesma função de forma iterativa

Função recursiva

- As funções recursivas geralmente tornam o programa mais legível e são definidas como na matemática, para evitar a retribuição de valores a variáveis
- Essas funções podem substituir trechos de código que envolvem laços de repetição, como os laços de repetições While e For
- Dizemos que uma função é recursiva quando dentro do corpo de uma função se faz uma chamada para a própria função

Algoritmo sem função recursiva

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      int inicio, limite;
7
8      inicio = 1;
9      limite = 21;
10
11     for(int i = inicio; i<limite; i++){
12         printf("%d ",i);
13     }
14     printf("\n\n");
15     system("pause");
16     return 0;
17 }
```

Algoritmo sem função recursiva

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Pressione qualquer tecla para continuar. . .
```

Algoritmo com função recursiva

```c
#include <stdio.h>
#include <stdlib.h>

int imprimeN (int inicio, int sfim);

int main()
{
    int comeco, sfim, i;

    comeco = 1;
    sfim = 21;

    printf("FUNCAO ITERATIVA\n");
    for(i = comeco; i < sfim; i++){     // impressao com for
        printf("%d ",i);    // imprime os numeros
    }
    printf("\n\n");
```

Algoritmo com função recursiva

```c
    printf("FUNCAO RECURSIVA\n");
    imprimeN(comeco, sfim);     //chamada da funcao recursiva

    printf("\n\n");

    system("pause");
    return 0;
}
int imprimeN (int comeco, int sfim){    //Função recursiva
    if(comeco<sfim){
        printf("%d ", comeco);      // imprime os numeros
        imprimeN(comeco+1, sfim);   //chamada recursiva
    }
}
```

Algoritmo com função recursiva

```
FUNCAO ITERATIVA
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

FUNCAO RECURSIVA
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

Pressione qualquer tecla para continuar. . .
```

Função recursiva com vetor

- **É quando usamos a chamada de uma função passando um elemento de um *array* como parâmetro e, depois, dentro dessa função, fazemos uma nova chamada para ela mesma**
- **Isto é, funciona do mesmo modo que uma função recursiva simples, porém, em vez de o parâmetro ser uma variável, agora será um vetor**

Função com vetor

```c
#include <stdio.h>
#include <conio.h>
#define max 5 //definindo uma constante


main(){
    int vet[max];
    int i, j;

    for (i=0;i<max;i++){

        printf("Digite o numero para o vetor [%d]: ", i);
        scanf("%d",&vet[i]); //preenchendo o vetor com dados
    }
```

Função com vetor

```
16      printf("\n\n");
17
18      for(j = 0; j< max; j++){
19          printf("Voce digitou o numero %d para o vetor [%d]\n",vet[j],j);
20
21      }
22
23      printf("\n\n");
24      system("pause");
25  }
```

Função com vetor

```
Digite o numero para o vetor [0]: 11
Digite o numero para o vetor [1]: 12
Digite o numero para o vetor [2]: 13
Digite o numero para o vetor [3]: 14
Digite o numero para o vetor [4]: 15


Voce digitou o numero 11 para o vetor [0]
Voce digitou o numero 12 para o vetor [1]
Voce digitou o numero 13 para o vetor [2]
Voce digitou o numero 14 para o vetor [3]
Voce digitou o numero 15 para o vetor [4]


Pressione qualquer tecla para continuar. . .
```

Função recursiva com vetor

```
1   #include <stdio.h>
2   #include <conio.h>
3   # define max 5 //definindo uma constante
4
5   int j=-1;
6   void exibir(int vetor[]);
7
8   main(){
9       int vet[max];
10      int i;
11
12      for (i=0;i<max;i++){
13
14          printf("Digite o numero para o vetor [%d]: ", i);
15          scanf("%d",&vet[i]); //preenchendo o vetor com dados
16      }
```

Função recursiva com vetor

```
18      printf("\n\n");
19
20      exibir(vet); //chamada da função recursiva
21
22      printf("\n\n");
23      system("pause");
24  }
```

Função recursiva com vetor

```
25
26  void exibir(int vetor[]){
27      j++;
28      if (j < max){ // condição de parada
29          //impressão do resultado da função.
30          printf("Voce digitou o numero %d para o vetor [%d]\n",vetor[j],j);
31          exibir(vetor); //chamada recursiva
32      }
33  }
```

Função recursiva com vetor

```
Digite o numero para o vetor [0]: 11
Digite o numero para o vetor [1]: 12
Digite o numero para o vetor [2]: 13
Digite o numero para o vetor [3]: 14
Digite o numero para o vetor [4]: 15


Voce digitou o numero 11 para o vetor [0]
Voce digitou o numero 12 para o vetor [1]
Voce digitou o numero 13 para o vetor [2]
Voce digitou o numero 14 para o vetor [3]
Voce digitou o numero 15 para o vetor [4]


Pressione qualquer tecla para continuar. . .
```

Funções macro – diretiva #define

- **Basicamente, uma diretiva avisa o compilador que ele deve procurar todos os eventos de determinada expressão e substituí-la por outra na compilação do programa**
- **Isso permite criar o que chamamos de funções macro**

- **Uma função macro é um tipo de declaração de função em que são informados o nome e os parâmetros da função como sendo o nome da macro e o trecho de código semelhante a ser aplicado na substituição**

- **A diretiva #define associa um identificador a uma cadeia de caracteres de *token***
- **Após a definição da macro, o compilador pode substituir a cadeia de caracteres de *token* em cada ocorrência do identificador no arquivo de origem**

A diretiva #define permite três sintaxes

1. Primeira sintaxe #define nome_da_macro

- **Nesta sintaxe, a diretiva define um nome que será usado em alguma estrutura do código**
  - **Exemplo: nome para ser testado em estruturas condicionais**

Algoritmo sem diretiva #define

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5      #ifdef status   // ifdef diretiva de compilação condicional
6          printf("O Status existeeeee!!!! Uhuuuuu!!!\n\n");
7      #else
8          printf("Status NAO definido. O #define FOI DECLARADO?\n\n");
9      #endif
10
11     system("pause");
12     return 0;
13 }
14
```

Algoritmo sem diretiva #define

```
Status NAO definido. O #define FOI DECLARADO?

Pressione qualquer tecla para continuar. . .
```

Algoritmo sem diretiva #define

```c
#include <stdio.h>
#include <stdlib.h>

#define status

int main() {
    #ifdef status   // ifdef diretiva de compilação condicional
        printf("O Status existeeeee!!!! Uhuuuuu!!!\n\n");
    #else
        printf("Status NAO definido. O #define FOI DECLARADO?\n\n");
    #endif

    system("pause");
    return 0;
}
```

Algoritmo sem diretiva #define

```
O Status existeeeee!!!! Uhuuuuu!!!

Pressione qualquer tecla para continuar. . .
```

A diretiva #define permite três sintaxes

2. **Segunda sintaxe #define nomeConstante valorConstante**
   - **Nesta sintaxe, a diretiva define um valor para uma constante que será usada ao longo do desenvolvimento do algoritmo**

Algoritmo com diretiva #define

```c
#include <stdio.h>
#include <stdlib.h>

#define PI 3.1415

int main() {

    printf("O valor de PI %.2f\n\n", PI);

    system("pause");
    return 0;
}

```

Algoritmo com diretiva #define

```
O valor de PI 3.14

Pressione qualquer tecla para continuar. . .
```

A diretiva #define permite três sintaxes:

3. **Terceira sintaxe #define nome_da_macro(PARÂMETROS) expressão**
   - **Nesta sintaxe se define uma função macro, e essa função macro é um pedaço de código pelo qual foi atribuído a um nome**

Algoritmo com diretiva #define

```c
#include <stdio.h>
#include <stdlib.h>

#define maior(x,y) x>y?x:y

int main() {
    int a = 12;
    int b = 6;
    int c = maior(a,b);

    printf("Maior valor = %d\n\n",c);

    system("pause");
    return 0;
}
```

Algoritmo com diretiva #define

```
Maior valor = 12

Pressione qualquer tecla para continuar. . .
```