1. Funções Recursivas

Qualquer função em linguagem C/C++ pode ser chamada de um modo recursivo, isto é, uma função pode chamar-se a si própria. O número de vezes que uma função pode ser chamada recursivamente é limitado ao tamanho da pilha (stack), se este valor for alcançado, o programa termina com "error: Stack Fault" ou "Stack Overflow".

Definição:

Função Recursiva é uma função que invoca a si própria, ou invoca outra função, e na sequência das diversas subfunções, uma das subfunções invoca a primeira função.

Cada vez que uma função é chamada de forma recursiva, é alojado e guardado uma cópia dos seus parâmetros por forma a não perder os valores dos parâmetros das chamadas anteriores. Em cada instância da função, só são directamente acessíveis os parâ etros criados para esta instância, não sendo directamente acessíveis os parâmetros de outras instâncias.

A informação guardada na invocação de uma função é designada por **Estrutura de Invocação** e consiste basicamente na seguinte informação:

- Endereço de retorno (quando a função terminar o programa deve continuar a sua execução na linha seguinte à invocação da função)
- Estado dos registos e flags do CPU
- Variáveis passadas como argumentos para a função (por valor, referência, etc.)
- Variável de retorno (por valor, referência, etc.)

A invocação de uma função recursiva é igual à invocação de uma função não recursiva, na qual é necessário guardar uma estrutura de invocação, sendo esta estrutura libertada depois do fim da execução da função e afectação/actualização do valor de retorno.

Características das funções recursivas:

As funções recursivas contêm duas partes fundamentais:

1. **Ponto de Paragem**: o ponto de paragem da recursividade é resolvido sem utilização de recursividade, sendo este ponto geralmente um limite superior ou inferior da regra geral.
2. **Regra Geral**: o método geral da recursividade reduz a resolução do problem através da invocação recursiva de casos mais pequenos, sendo estes casos mais pequenos resolvidos através da resolução de casos ainda mais p equenos, e ass sucessivamente, até atingir o ponto de paragem que finaliza o método.

Exemplo:

$$Factorial(n) = \begin{cases} (n == 0) \rightarrow 1 & \text{Ponto de Paragem} \\ (n) \rightarrow n * (n-1)! & \text{Regra Geral} \end{cases}$$

Nota:

As variáveis declaradas como estáticas, não requerem novo armazenamento, em cada invocação da função recursiva. Estas variáveis existem durante todo o programa, não sendo criado cópias destas variáveis na invocação de uma função de modo recursivo.

Desenho de algoritmos recursivos:

1. Obter uma solução de como o problema pode ser dividido em passos menores
2. Definir um ponto de parage
3. Definir uma regra geral
4. Verificar se o algoritmo termina
5. Desenhar uma árvore de execução do programa, mostrando o desenvolvimento do processo.

Utilizar recursividade ou não ?

- Na fase de concepção, os métodos recursivos são um bom processo flexível de pensamento
- Na fase de imp ementação, é necessário avaliar os diversos métodos e questionar qual o melhor método para as circunstâncias especificas (mais eficaz e eficiente).

Conclusão:

A recursividade deve ser utilizada quando a forma recursiva é a forma mais simples e intuitiva de implementar uma solução para a resolução do problema.

Exemplo n.º 1

Escrever uma sequência de números na vertical

```
#include <iostream.h>

void esc_numero1(int num)
{
 if (num > 0)
 { esc_numero1(num-1);
   cout << num << " ";
 }
}

void esc_numero2(int num)
{
 if (num > 0)
 { cout << num << " ";
   esc_numero2(num-1);
 }
}

int main()
{
 int numero=0;

 cout << "Introduza o numero inicial (>0)";
 cin >> numero;
 esc_numero1(numero);
 cout << endl;
 esc_numero2(numero);
}
```

Exemplo n.º 2

Função Factorial

Método Recursivo

```
long int factorial1( int num )
{
     if (num < 0)
      {  // Se o número é menor que zero então retornar erro.
        retun(-1);
      }
     if ( (num == 0) || (num == 1) )
      {  // Se o numero é igual a 0 ou 1, então o seu factorial é 1.
       return(1);
      }
     else
      {  // Caso contrário, invocar recursivamente a função.
       return( num * factorial1( num - 1 ) );
      }
}
```

Método Iterativo

```
long int factorial2( int num )
{ long int resultado = 1;

     if (num < 0)
      {  // Se o número é menor que zero então retornar erro.
        retun(-1);
      }
     for(int i = 2; i <= num; i++)
      { resultado *= i; }
     return(resultado);
}

void main()
{
     int numero=0;
     long int fact=0;

     cin >> numero;
     fact = factorial1(numero);
     cout << endl << numero << "!= " << fact << endl;
     fact = factorial2(numero);
     cout << endl << numero << "!= " << fact << endl;
}
```

Nota:

A função factorial retorna o mesmo valor em ambas as implementações, contudo, a forma sem recursividade é ligeiramente mais eficiente.

Exemplo n.º 3

Inversão de uma string

Método Recursivo

```
void stringinv_aux (char * s, int i, int j)
{
     if ( j - i > 2)
      { stringinv_aux (s, i + 1, j – 1);  // Inverter a sub-string.
       }
     //  Trocar os caracteres extreinos.
     char aux = s[j];  s[j] = s[i]; s[i] = aux;
}

void stringinv(char * str)
{
     stringinv_aux(str, 0, strlen(str));
}
```

exemplo: Inverter a string “bonitos”

instância	i	j	string à entrada da função	string à saída da função
1	0	6	bonitos	sonit
2	1	5	sonit	sonit
3	2	4	sonit	sotinob

Método Iterativo

```
void stringinv_aux(char * str, int i, int j)
{
     for( int i = O ; i < j ; ++i, --j )
      {  //  Trocar os caracteres extremos.
         char aux = s[j]; s[j] = s[i]; s[i] = aux;
      }
}

void stringinv(char * str)
{
     stringinv_aux(str, 0, strlen(str));
}
```

Exemplo n.º 4

Converter números inteiros em strings

Método Recursivo

```
void itoa_aux(int num, char * &s)
{
     if ( num == 0 ) return ;
     itoa_aux( int(num / 10), s);
     *s++ = n % 10 + '0';
}

void itoa(int num, char *s)
{
     if (num < 0) // Se o valor for negativo.
      { *s++ = '-';
        n = -n;
      }
     itoa_aux( num , s );
     *s = 0;       // Terminar a String
)
```

Método Iterativo

```
void itoa(int num, char * s)
{ int valor, i = 0, j;
  char c;

     if  ( (valur = num) < 0)
      { num = -num;
      }
     do
      { s[i++]= n %10 + '0';
      }
     while(( num /= 10) > O );

     if ( valor < 0)
      { s[i++] = '-'; }
     s[i] = '\0';

     // Inverter a string
     for ( j = 0, --i; i > j ; --i, ++j)
     { c = s[i];  s[i] =s[j]; s[j] = c; }
 }
```

Exemplo n.º 5

Cálculo do resto da divisão de dois números

Considere a seguinte função que calcula o resto da divisão de dois números, através da sucessiva subtracção ao 1º número do valor do 2º número.

Método Recursivo

```
#include <iostream>
int res(int numerador, int denominador);

int res(int x, int y)
{
 if (x < y) return(x);

 return( res(x – y, y) );
}
```

Método Iterativo

```
#include <iostream>
int res(int numerador, int denominador);

int res(int x, int y)
{
  while(x >= y)
  { x = x –y;
  }
  return( x );
}
```

Exemplo n.º 6

Cálculo do resto da divisão de dois números

Considere a seguinte função que calcula a multiplicação de dois números positivos, maiores qu zero, através de sucessivas somas do primeiro número.

Método Recursivo

```
#include <iostream>
long int mult(int x, int y);

long int mult(int x, int y)
{
 if (y == 1)
   return(x);
 else
   return(x + mult(x, y-1));
}
```

Método Iterativo

```
#include <iostream>
long int mult(int x, int y);

long int mult(int x, int y)
{  long int res=0;
   while( y != 1)
   { res += x;
     y--;
   }
   return(res);
}
```

Exemplo n.º 7

Cálculo da seguinte formula:

$$\sum_{i=1}^{n} (i * i)$$

Método Recursivo

```
#include <iostream>
long int sumatorio(int num);

long int sumatorio (int num)
{
 if (num == 0)
      return(0L);
 else
      return( num * num + sumatorio( num –1 );
}
```