Funções

- Recursividade (continuação)

Ainda podemos definir uma função recursiva que implemente a operação de multiplicação com base na operação de adição:

```
int multiplicar (int A, int B)
{
  if (!B)
    return (0);
  else
    return ( A + multiplicar (A, B-1) );
}
```

Funções

- Recursividade (continuação)

Para uma melhor compreensão do que foi apresentado, devemos nos recordar do conceito de “parâmetros formais” e definirmos o que vem a ser “registro de ativação”.

O registro de ativação é uma área de memória que guarda informações referentes ao estado atual de uma função:

- valor dos parâmetros formais;
- valor das variáveis locais;
- valor do contador de programa (PC);
- etc.

- Recursividade (continuação)

Sempre que uma função chama outra função o registro de ativação da função invocadora é salvo e um novo registro de ativação é criado para a função invocada.

Este processo é conhecido como salvamento e troca de contexto e pode ser melhor compreendido se o aplicarmos sobre um programa que se utilize da função recursiva multiplicar definida anteriormente.

```
#include <stdio.h>
int multiplicar (int A, int B);
int main()
{
      int a, b, resp;
      printf("\nMultiplicando: ");
      scanf("%d", &a);
      printf("\nMultiplicador: ");
      scanf("%d", &b);
      resp = multiplicar (a, b);
      printf("\n%d * %d = %d\n",a,b,resp);
      return 0;
}
int multiplicar (int A, int B)
{
      if (!B)
        return (0);
      else
        return ( A + multiplicar (A, B-1) );
}
```

Funções

- Recursividade (continuação)

A execução de todo programa escrito na linguagem C começa pela execução da função main(). Logo, o registro de ativação desta função é gerado, nele são armazenados (no caso do nosso exemplo) os valores das variáveis locais a, b e resp, e a informação de qual instrução da respectiva função encontra-se sendo executa. O programa segue sua execução até que na linha “resp = multiplicar (a, b);” é efetuada uma chamada a função multiplicar. Neste ,momento o registro de ativação da função main() é salvo (para posteriormente o contexto atual poder ser restaurado) e um novo registro de ativação é gerado para armazenar informações referentes a função multiplicar.

Funções

- Recursividade (continuação)

O registro de ativação da função multiplicar conterá os valores armazenados nas variáveis locais A e B, e a informação de qual instrução da respectiva função encontra-se sendo executa. A primeira instrução do corpo da função multiplicar() é uma instrução condicional, a qual verificará se a função multiplicar deve ou não chamar a si mesma, caso esta verificação resulte em falso, o registro de ativação da função multiplicar, atualmente sendo executada, será salvo e um novo registro de ativação será criado para a nova instância da função multiplicar(), este processo se dará até que a função multiplicar deixe de chamar a si mesma.

Funções

- Recursividade (continuação)

Com base no que foi exposto, podemos compreender uma das desvantagens da utilização de recursividade, que seria?

O consumo de memória necessário para a troca de contexto.

Exercício:

Para uma melhor compreensão do conceito de recursividade faça agora uma função recursiva, na linguagem C, para calcular o fatorial de um número natural e construa um programa que a utilize.

Funções

Um outro exemplo muito utilizado de problema que possui uma definição recursiva é a geração série de Fibonacci:

{0,1,1,2,3,5,8,13,21,34, …}

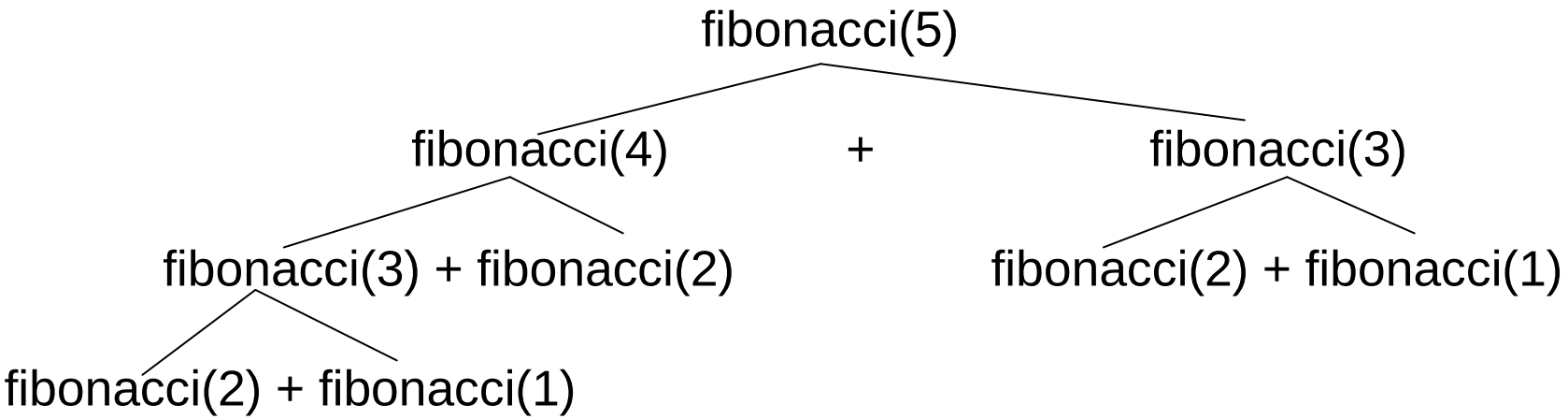
Uma função recursiva que recebe a posição do elemento na série e retorna seu valor é:

```
unsigned int fibonacci(unsigned int i)
{
  if (i==1)
     return 0;
  if (i==2)
     return 1;
  return (fibonacci(i-1) + fibonacci(i-2))
}
```

Funções

Fora o problema do consumo de memória, gerado pela troca de contexto na recursão, qual seria outro problema proveniente da recursão evidenciado na função recursiva apresentada para o cálculo do valor de um elemento da série de Fibonacci com base na sua posição?

O cálculo do mesmo valor n vezes.



Funções

Mesmo problemas que possuem uma definição recursiva sempre podem ser solucionados de forma imperativa. Um exemplo disso é o cálculo do valor de um elemento da série de Fibonacci com base na sua posição imperativo através da função abaixo:

```
unsigned int fibonacci(unsigned int i)
{
   if (i==1)
      return 0;
   if (i==2)
      return 1;
   else
   {
       unsigned int a, b;
       for(a=0, b=1; i-2; b+=a,a=b-a,i--);
       return b;
   }
}
```