Universidade de Brasília

Recursão

Algoritmos e Estruturas de Dados

Prof. Daniel Guerreiro e Silva

Roteiro

1. Definição
2. Implementações

Leitura sugerida: Seções 5.1 a 5.4 e 5.8 do livro-texto (Drozdek)

Técnicas de programação

- A recursão é uma técnica útil e elegante de se usar no projeto de algoritmos

- A idéia de recursão remonta à noção de definir objetos e conjuntos através de **autorreferências**.

Definição recursiva

- Problemas em que uma dada instância contém uma instância menor **dele mesmo** podem ser abordados de forma recursiva.
- Nesse sentido, uma definição recursiva consiste de duas partes:
  - Um **caso base** onde se listam os elementos básicos que formam todos os outros elementos do conjunto / problema
  - Um **conjunto de regras** que permite construir novos elementos a partir dos elementos básicos ou anteriormente construídos

Exemplo: fatorial

$$n! = \begin{cases} 1 & se\ n = 0 \\ n \cdot (n-1)! & se\ n > 0 \end{cases}$$

- Podemos escrever uma função em C++ que explore esta definição recursiva.

Exemplo: fatorial

```
long int fatorial(int n) {
     if(n == 0)
          return 1;
     else {
          return fatorial(n-1) * n;
     }
}
```

Como a recursão funciona

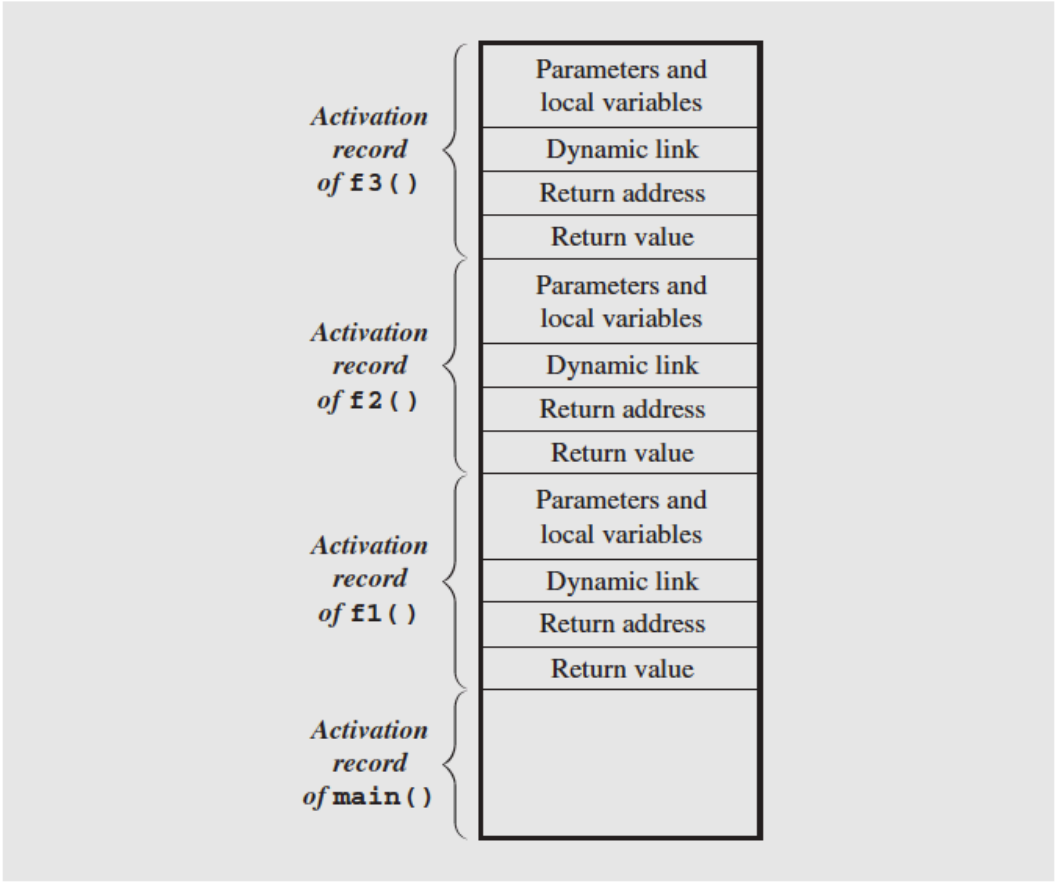
- Funções recursivas são implementadas por meio de uma **pilha em tempo de execução**, gerenciada pelo sistema operacional
- Quando uma função é chamada:
  - Os parâmetros são inicializados com os valores passados
  - O sistema deve saber onde reiniciar a execução do programa, quando a função terminar
- Informações como estas são armazenadas na pilha de execução do programa.

Como a recursão funciona

- Toda função, inclusive a `main()`, contém informações do seu estado em um elemento chamado **registro de ativação:**
  - Parâmetros e variáveis locais
  - Endereço de retorno da função
  - Valor de retorno (funções não-void)
  - Vínculo dinâmico (ponteiro para registro de ativação do ativador)
- Cada vez que se chama uma função, um registro de ativação é criado e colocado na pilha de execução

Como a recursão funciona

- Suponha que `main()` chame `f1()`, que chame `f2()` e que finalmente chame `f3()`
- Observe ao lado a pilha durante a execução de `f3()`



Como a recursão funciona

- O término de uma função implica na remoção do seu registro de ativação da pilha de execução do programa.
- Se uma função chamar ela mesma, o que na prática ocorre é a instanciação de uma função chamando outra instanciação do mesmo código, porém com registros de ativação distintos!
- Logo a recursão funciona por meio deste mecanismo

Como a recursão funciona

1.main()
2. fatorial(3)
3. fatorial(2)
4. fatorial(1)
5. fatorial(0)
5. 1
4. 1 * 1 = 1
3. 2 * 1 = 2
2. 3 * 2 = 6
1.6

Implementações

Exemplo: sequência de Fibonacci

- A sequência de Fibonacci é dada pelos inteiros
  $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144\ldots$

- Inicia-se com os números 0 e 1, note daí que o terceiro termo em diante é a soma dos dois anteriores.
- Logo podemos escrever a relação de recorrência do n-ésimo termo de Fibonacci como

$$f(n) = \begin{cases} n & n = 0, n = 1 \\ f(n-1) + f(n-2) & caso\ contrário \end{cases}$$

Exemplo: sequência de Fibonacci

```
long int fib(int n) {
     if(n<=1)
          return n;
     else  {
          return fib(n-1) + fib(n-2);
     }
}
```

Funções recursivas X não-recursivas

- Funções recursivas em geral levam a soluções de código simples e mais elegantes.
- Porém, um código recursivo:
  - é mais complexo de depurar
  - pode causar uso desnecessário do espaço da pilha de execução
  - pode causar um número excessivo de chamadas de função
- Embora não seja necessariamente fácil, pode-se obter uma implementação não-recursiva para toda função recursiva

Fibonacci não-recursivo

```
long int fibo(int n) {
     int f1=0, f2=1, f3, i;
     for(i=0; i<n; i++) {
          f3 = f1 + f2;
          f2=f1;
          f1=f3;
     }
     return f1;
}
```

Fatorial não-recursivo

```
long int factorial(int n) {
  long int product = 1;
  while (n>0) {
    product *= n;
    --n;
  }
  return product;
}
```

Comparativo

- Veja o código recursao.cpp, recursao.h e compara_recursao.cpp

Conclusões

- A recursão é uma técnica poderosa de programação que leva ao projeto de algoritmos simples e eficientes
- Como veremos adiante, soluções elegantes para problemas como ordenação, busca, e percurso em árvores são obtidas com funções recursivas
- Entretanto, o seu uso deve ser feito com critério e, nos casos de linguagens procedurais / imperativas (C, C++, Java, ...), soluções iterativas, se factíveis, devem ser priorizadas