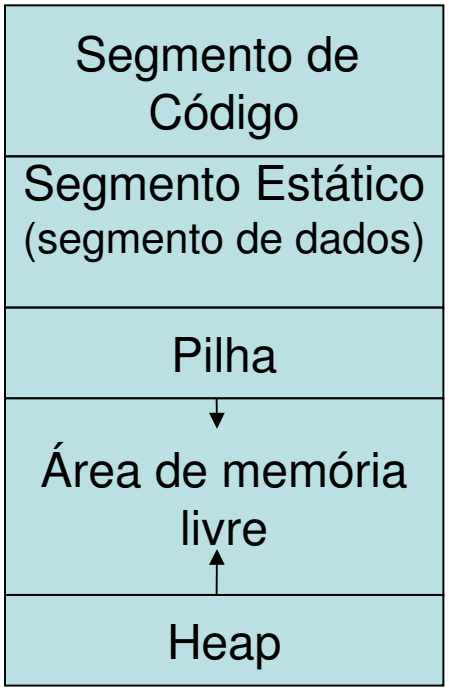
Ambiente de tempo de Execução
Cap. 7

Introdução

- Subdivisão da memória usada pelo programa em tempo de execução



Introdução

- A fim de preparar a geração de código, deve-se relacionar o fonte estático do programa às ações em tempo de **execução**.
- Durante a execução, o mesmo nome no código fonte pode estar associado a diferentes objetos de dados na máquina alvo.

Introdução

- A alocação e liberação de objetos de dados é gerenciado pelo ambiente de tempo de execução (runtime environment)
- Cada **execução** de um procedimento é referenciada como uma **ativação**.
- No caso de chamadas recursivas, várias ativações de um mesmo procedimento estão “vivas” em determinado instante.

Introdução

- A representação de um objeto de dados em tempo de execução é determinado pelo seu tipo.
- Freqüentemente, tipos de dados elementares, tal como caracteres, inteiros e real, podem ser representados por objetos de dados equivalentes na máquina alvo.

Procedimentos

- Procedimentos possuem
  - **Definição** – consiste na declaração do procedimento formando por um **nome**
  - **Corpo** – é o enunciado do procedimento
  - **Parâmetros formais** – são os identificadores que aparecem na definição de um procedimento (argumentos)
  - **Parâmetros atuais** – são passados ao procedimento durante uma chamada

Procedimentos

- **Funções** são procedimentos que retornam valores
- Um procedimento é **chamado** quando seu nome aparece dentro de um enunciado executável
- Assim que o procedimento é chamado, seu corpo é executado
- O fluxo de controle sequencial dos procedimentos pode ser exibido na forma de **árvores de ativação**

Procedimentos

- Suposições sobre o fluxo de controle entre procedimentos:
  - O controle flui seqüencialmente; isto é, a **execução** de um programa consiste em uma seqüência de passos, com o controle, a cada passo, em algum ponto específico do programa;
  - Cada **execução** de um procedimento começa ao início do corpo do procedimento e eventualmente retorna o controle para o ponto imediatamente seguinte ao local de onde foi chamado.

Árvores de Ativação

- **Ativação** é a execução do corpo de um procedimento que possui um ***tempo de vida*** específico
- O tempo de vida de uma **ativação** é a seqüência de passos entre o primeiro e o último passos na **execução** do corpo do procedimento, incluindo o tempo utilizado para a **execução** de procedimentos chamados.

Árvores de Ativação

- Pode ser controlado sinalizando o momento em que se entra e sai de um procedimento
- Procedimentos **recursivos** iniciam uma ativação antes que uma anterior tenha sido encerrada

Árvores de Ativação

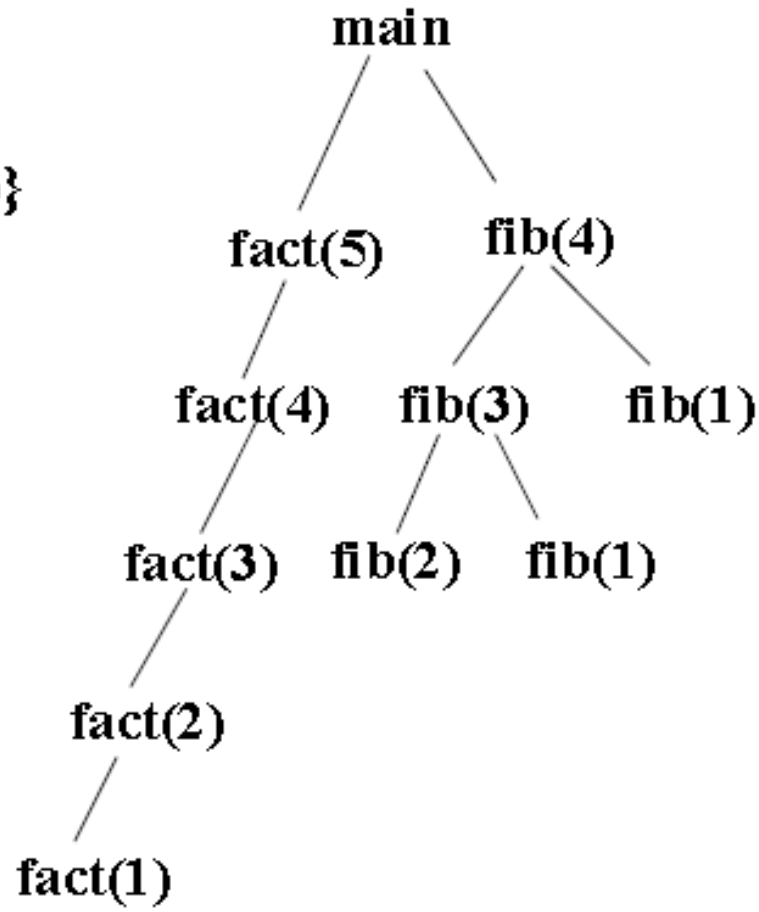
- Se *a* e *b* são ativações de procedimentos, então seus tempos de vida ou são não sobrepostos ou são aninhados.
  - Isto é, se a **ativação** *b* é iniciada antes de *a* ser deixada, o controle deve abandonar *b* antes de deixar *a*.

Árvore de Ativação

- Em uma ***árvore de ativação***
  - Cada nó é uma ativação de um procedimento
  - A raiz é a ativação do programa principal
  - O nó para ***a*** é o pai do nó para ***b*** se e somente se o controle fluir da ativação de ***a*** para ***b***
  - Dado ***a*** e ***b*** nós irmãos. O nó ***a*** está a esquerda do nó ***b*** se e somente se o tempo de vida de ***a*** terminar antes do tempo de vida de ***b***

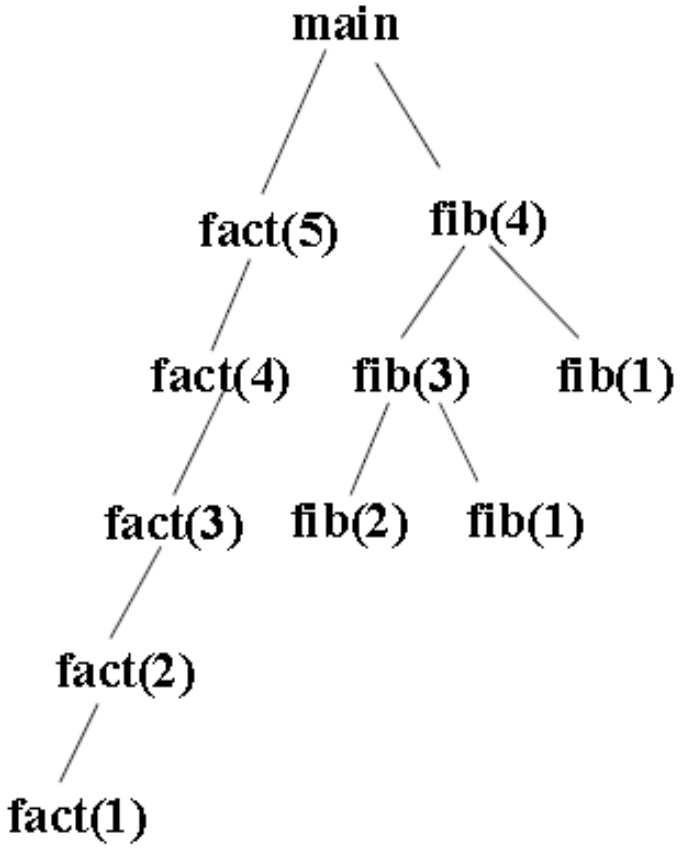
Árvores de ativação - Exemplo

```
main
{   fact(x){…fact(x-1)}
    fib(y) { … fib(y-1)+fib(y-2)}
    fact(5)
    fib(4)
}
```



DFS da árvore de ativação

- O fluxo de controle em um programa corresponde a uma travessia em profundidade (depth-first search DFS) da árvore de ativações
- Seqüência de chamadas corresponde a um percorrimento em pré-ordem
- Seqüência de retornos corresponde a um percorrimento em pós-ordem



Pilha de Controle

- Em uma **pilha** de controle, o nó é empilhado para uma **ativação** quando esta iniciar, e é desempilhado quando a **ativação** terminar.
- O conteúdo da **pilha** de controle está relacionado a percursos até a raiz da árvore de ativações.

Escopo de uma declaração

- Escopo de uma declaração:
  - A parte do programa ao qual uma declaração se aplica chama-se **escopo**
- As **regras de escopo** determinam a declaração de nomes
  - **Local**: um nome ocorre dentro de um procedimento
  - **Global ou não-local**: um nome que ocorre fora de um procedimento

Amarração de nomes

- Um **objeto de dados** corresponde a uma localização de memória
- Ambiente mapeia o nome ao objeto de dados
- Estado mapeia o objeto de dado ao valor armazenado
- Atribuições modificam o estado, mas não o ambiente
- Um **nome** *x* está ***amarrado*** a um **objeto de dados** *s* quando existe uma associação entre *x* e *s*

Questões relacionadas à organização da memória e amarração de nomes

- Os procedimentos são recursivos?
- O que acontece aos valores dos nomes locais quando o controle retorna da **ativação** de um procedimento?
- Um procedimento pode se referir a nomes não-locais?
- Como são transmitidos os parâmetros quando um procedimento é chamado?

Questões relacionadas à organização da memória e amarração de nomes

- Os procedimentos podem ser transmitidos como parâmetros?
- Os procedimentos podem ser retornados como resultados?
- A **memória** pode ser reservada dinamicamente sob controle do programa?
- A **memória** precisa ser liberada explicitamente?

Memória em tempo de execução

- A maneira como um compilador organiza a memória para amarrar nomes determina
  - Suporte a procedimentos recursivos
  - Controle do tempo de vida das variáveis locais
  - Regras de referência a nomes não-locais
  - Passagem de parâmetros em chamada de procedimentos
  - Passagem de procedimento como parâmetro
  - Alocação dinamica da memória

Memória em tempo de execução

- Em geral, um programa compilado obtém um ***bloco de memória*** do sistema operacional para ser executado
- Este bloco de memória costuma ser dividido em
  - Seção do código alvo gerado
  - Seção de objetos de dados
  - Pilha de controle para ativação de procedimentos

Memória em tempo de execução

- O tamanho do código alvo é estático e estabelecido em tempo de compilação
- O tamanho de alguns objetos de dados também é estático, determinado em tempo de compilação
- O controle das chamadas de procedimentos é controlada por uma ***pilha***
- Outras informações dinâmicas são alocadas em uma área livre denominada ***heap*** que cresce para cima

Memória em tempo de execução

- Quando ocorre uma chamada, a **execução** de uma **ativação** é interrompida e as informações sobre o estado da máquina, tais como o valor do apontador da próxima instrução e dos registradores de máquina são salvas na **pilha**.

Registros de Ativação

- As informações necessárias em uma única **execução** de um procedimento são gerenciadas utilizando-se um único bloco contíguo de armazenamento chamado "**registro** de **ativação**"

Registros de Ativação

- Um procedimento precisa registrar as seguintes informações para ser executado
  - **Valor retornado** – contém o valor retornado pela função
  - **Parâmetros atuais** – contém o valor dos parâmetros atuais
  - **Elo de controle opcional** – aponta para o registro de ativação do procedimento que o chamou
  - **Elo de acesso opcional** – para acessar dados de outros registros de ativação

Registros de Ativação

- Um procedimento precisa registrar as seguintes informações para ser executado (cont.)
  - **Estado da máquina** – condição da máquina antes de executar o procedimento, para o qual deve retornar
  - **Dados locais** – variáveis locais ao procedimento
  - **Temporários** – resultados de expressões

Layout dos Dados Locais em Tempo de Compilação

- Cada **tipo** determina a quantidade de *bytes* que deve ser reservada a um nome
- A disposição de memória para objetos de dados é influenciada pelas restrições de endereçamento da máquina-alvo
- Por isto, é desejável que as instruções que operem sobre dados do mesmo tipo estejam ***alinhadas***
- Para garantir o alinhamento, pode ser necessário inserir espaços sem uso chamados ***enchimento***

Estratégias de alocação de memória

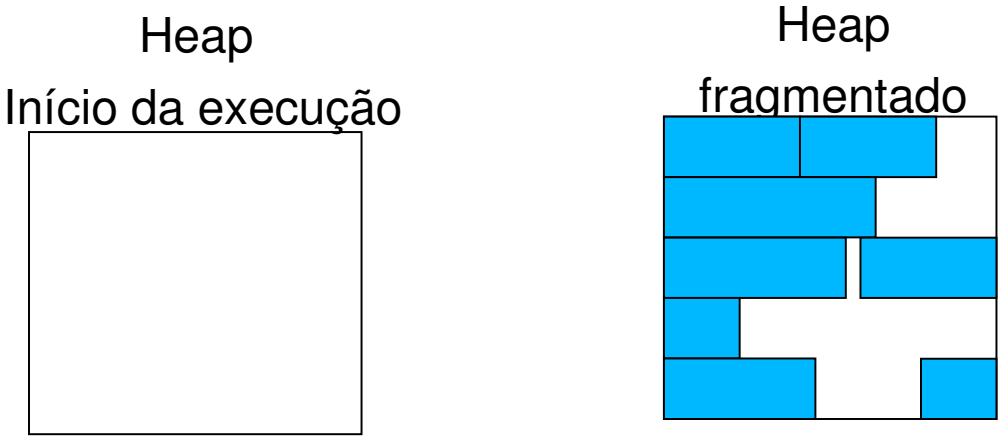
- O projeto de compilador deve incluir estratégias de alocação de memória em cada uma das áreas de dados
  - **Alocação de memória estática** – não precisa de suporte em tempo de execução
  - **Alocação de memória de pilha** – exige suporte para empilhar/desempilhar registros de ativação
  - **Alocação de heap** – suporte para alocação contígua e liberação aleatória de memória em tempo de execução

Gerenciamento do Heap

- Heap: Área da memória que armazena dados cujos tamanhos e os tempos de vida são indefinidos
- Gerenciador de memória: aloca e desaloca espaço no heap. Além disso:
  - Eficiência de espaço, eficiência do programa, baixa sobrecarga
- Gerenciador deve controlar a fragmentação do heap

Gerenciamento do Heap

- No início, o heap está completamente vazio
- A medida que dados são alocados e desalocados, a fragmentação pode ocorrer

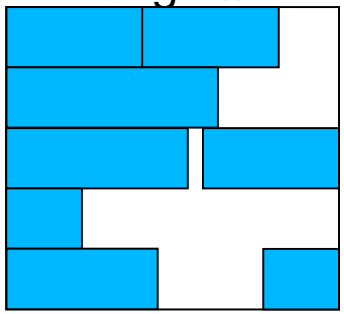


- Algoritmos para reduzir fragmentação
  - Best-fit: colocar um dado no espaço livre com menor perda
  - First-fit: colocar dado alocado no primeiro espaço livre com tamanho suficiente para armazenar o dado

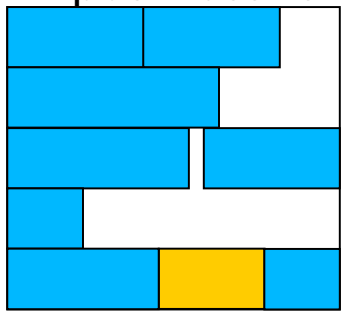
Gerenciamento do Heap

- Best-fit: exemplo

Heap original

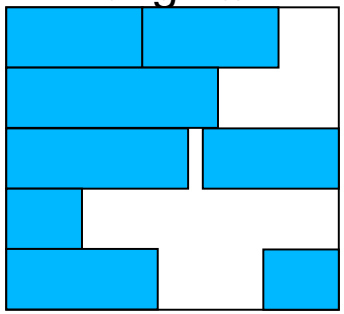


Heap Após Best-fit

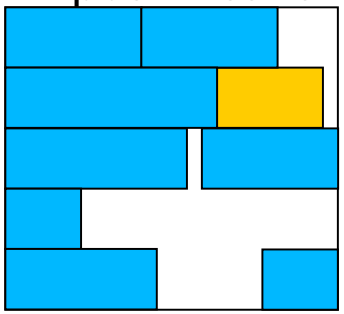


- First-fit:

Heap original



Heap Após First-fit



- Algoritmo?

Gerenciamento do Heap

- Desalocação manual (C, C++) é propensa a erros
  - Não desalocar dados não usados (*memory leak*)
  - Referenciar dados já desalocados (*dangling pointer-deference*)

- Algoritmos de garbage collection podem minimizar esses erros

Gerenciamento do Heap

- Exercício
  - Supor que o heap possui sete áreas livres (*chunks*) de 80, 30, 60, 50, 70, 20 e 40 bytes
  - *Chunks* < 8 bytes não são permitidos -> O *chunk* inteiro será ocupado
  - Objetos com tamanhos de 32, 64, 48 e 16 bytes (nessa ordem)
  - Como fica o espaço livre dos chunks, satisfazendo as restrições acima, com os algoritmos: best-fit e first-fit?

Coleta de lixo (*garbage collection*)

- Dados que não podem ser referenciados no programa -> garbage
- Técnicas de garbage collection são usadas desde 1958 (Lisp)
  - Linguagens que oferecem mecanismos de garbage collection: Java, ML, Modula-3, Prolog, Perl, Smalltalk

- Algoritmos para garbage collection
  - Contagem de referências
  - Marcar e varrer

Coleta de lixo (*garbage collection*)

- Contagem de referências
  - Todo objeto possui um campo para contagem de referências
  - Objeto é desalocado quando esse campo é zero
    - Na alocação de um objeto: contagem de referência é 1
    - Passagem de parâmetro: contagem de ref. é incrementado
    - Atribuição de referências: Na atribuição $u=v$, o objeto referenciado por v é incrementado e o objeto antigo, apontado por u é decrementado
    - Retorno de procedimento: todas as referências mantidas por variáveis locais do procedimento devem ser decrementadas
    - Transitividade: toda vez que a referência de um objeto v é zero, deve-se decrementar as referências de objetos apontadas por v

Coleta de lixo (*garbage collection*)

- Contagem de referências
  - Operações adicionais são executadas para cada atribuição de referências, entrada e saída de procedimentos
  - A sobrecarga desse mecanismo é O(NC), onde NC=número de computações do programa

Coleta de lixo (*garbage collection*)

- Algoritmo Marcar e Varrer
  - Passo 1) Marcar todos os objetos alcançáveis
  - Passo 2) Varrer o heap inteiro a fim de liberar objetos inalcançáveis
  - Entrada: conjunto raíz de objetos, heap, uma lista de *chunks* livres do heap
  - Saída: lista de *chunks* livres modificada

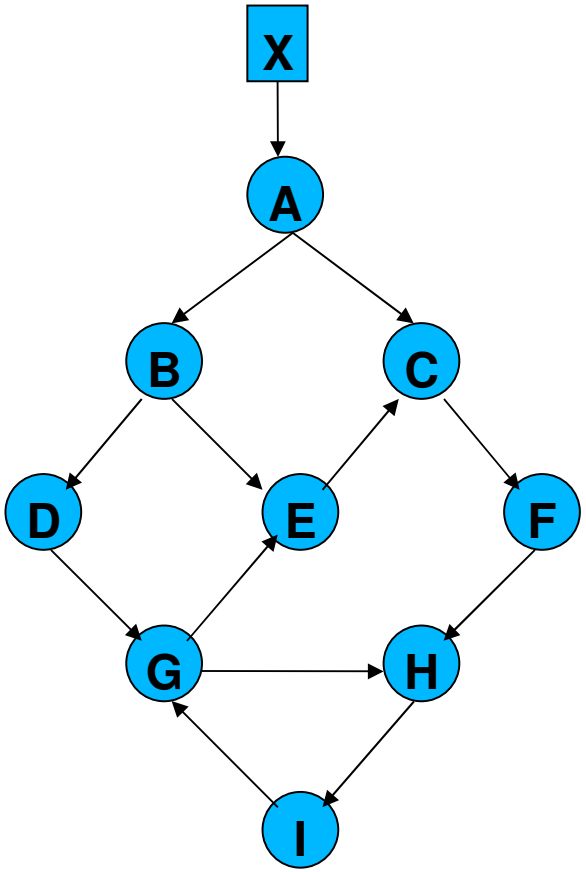
Coleta de lixo (*garbage collection*)

- Algoritmo Marcar e Varrer

2. Configurar *reached-bit*=1 de cada objeto no conjunto raíz
3. Adicionar os objetos do conjunto raíz para a lista *Unscanned*
4. While (*Unscanned* !=0)
5. remove um objeto *o* de Unscanned
6. for (each *o'* referenciado por *o*)
7. if ($o'_{reached\text{-}bit}$==0)
8. configurar $o'_{reached\_bit}$=1;
9. adicionar *o'* em *Unscanned*
10. Free={};
11. for (each chunk de memória *o* no heap)
12. if ($o_{reached\text{-}bit}$==0)
13. adicionar *o* para Free
14. else configurar $o_{reached\text{-}bit}$=0

Coleta de lixo (*garbage collection*)

- Exercícios



O que acontece com as contagens de referências dos objetos da Figura se:

- **A referência de A para B é excluída**
- **O objeto C é deletado**