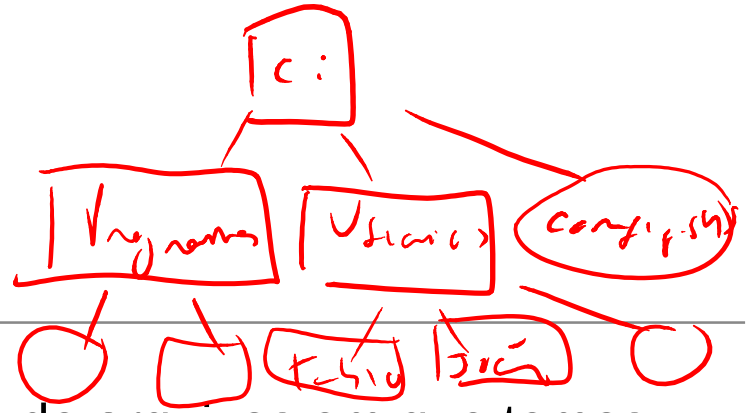
Introdução à Programação C

Fabio Mascarenhas - 2014.2

http://www.dcc.ufrj.br/~fabiom/introc

Um Sistema de Arquivos



- Suponha que queremos representar um sistema de arquivos em que temos arquivos e diretórios
- Um arquivo tem um nome e um tamanho
- Um diretório tem um nome, uma lista de arquivos, mas também tem uma lista de *outros diretórios*
- Quando a definição de um tipo de dado inclui outras instâncias desse mesmo tipo como parte dele dizemos que esse tipo de dado é *recursivo*

`struct Arq` e `struct Dir`

- Usamos vetores dinâmicos para as listas de arquivos e diretórios:

```
struct Dir {
    char nome[MAXNOME];
    int uso_arqs;
    int cap_arqs;
    struct Arq *arqs;
    int uso_dirs;
    int cap_dirs;
    struct Dir *dirs;
};
```

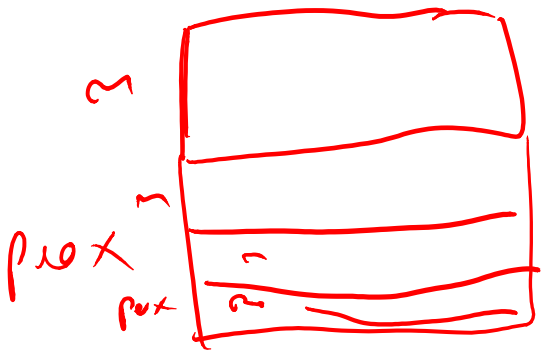
```
struct Arq {
    char nome[MAXNOME];
    int tamanho;
};
```

- Poderíamos usar um vetor de tamanho fixo para os arquivos (`struct Arq arqs[MAXARQS]`), mas não para o vetor de diretórios!

- Tente pensar em qual é o tamanho em memória de uma `struct Dir` acima, e qual seria ele se o campo `dirs` fosse um vetor de `struct Dir`

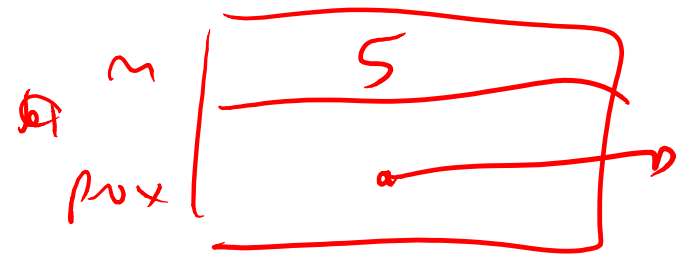
Campo recursivo

- Um campo recursivo sempre deve ser um ponteiro para a estrutura, e só pensar em seu *layout* na memória:



```
struct Rec {
    int n;
    struct Rec prox;
};
```

```
struct Rec {
    int n;
    struct Rec *prox;
};
```



sizeof(struct Rec) = sizeof(int) + sizeof(struct Rec)

X = 4 + X

sizeof(struct Rec) = sizeof(int) + sizeof(struct Rec*)

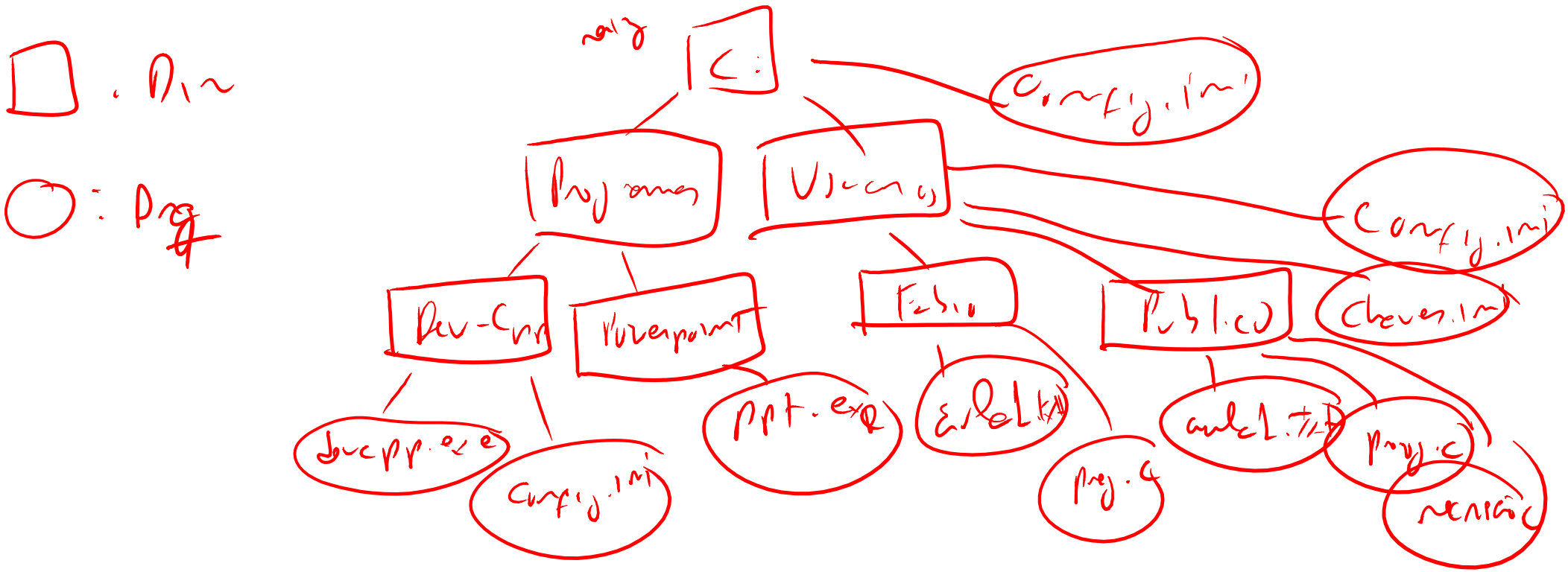
Criando diretórios

- Criar um novo diretório e adicionar um novo arquivo ou diretório a um já existente são tarefas simples
- Um novo diretório não tem arquivos nem outros diretórios
- Adicionar um arquivo a um diretório é só copiar seus dados para o vetor de arquivos, aumentando sua capacidade se necessário
- O mesmo para adicionar um diretório a outro

Sistema de arquivos

- Podemos representar o *sistema de arquivos* pelo seu *diretório raiz*

- Os diretórios formam uma *árvore* a partir desse diretório raiz



Espaço total

- Como podemos calcular o espaço total ocupado por todos os arquivos do nosso sistema?
- Para um arquivo é simples, é o seu tamanho
- Para uma lista de arquivos, basta somar todos os tamanhos
- E para um diretório?

kbytes_usados

- Definir a função que calcula quanto de espaço um diretório usa é fácil, basta usar a própria função para saber quando cada subdiretório dele usa:

```
static int kbytes_usados(struct Dir *dir) {
    int i;
    int total = 0;
    for(i = 0; i < dir->uso_arqs; i++) {
        total = total + dir->arqs[i].tamanho;
    }
    for(i = 0; i < dir->uso_dirs; i++) {
        total = total + kbytes_usados(&(dir->dirs[i]));
    }
    return total;
}
```

- Dizemos que a função kbytes_usados é uma função *recursiva*

Caso base, caso recursivo

- Uma função recursiva chama ela própria como parte de sua execução, mas nem sempre ela realmente vai fazer isso, ou o programa nunca terminaria!

- Quando ela consegue executar sem se chamar novamente, dizemos que ela atingiu um *caso base*

- Para `kbytes_usados`, o caso base é um diretório sem nenhum subdiretório

- Os outros casos são chamados de *casos recursivos*, e a ideia é que sempre que chamamos a função novamente estamos chamando ela em um problema "menor", e alguma hora chegaremos em um caso base

Procurando um diretório por nome

- Muitas coisas que podemos querer fazer com uma estrutura recursiva são implementados por funções recursivas

- Por exemplo, como poderíamos implementar uma função `busca_dir`, que recebe um ponteiro para um diretório e um nome, procura um diretório com aquele nome, e retorna um ponteiro para ele (ou 0 se não achar)?

- O *caso base* é o caso em que o nome que estamos procurando é o do próprio diretório passado pra `busca_dir`

- Para o caso recursivo, retornamos o primeiro resultado de `busca_dir` nos subdiretórios que não for 0

Recursão em mais de um parâmetro

- Uma chamada recursiva de uma função pode envolver vários parâmetros
- Pense em uma variante de `busca_dir` com a assinatura abaixo, que busca a enésima ocorrência do diretório com o nome dado

```
static struct Dir *busca_dir(struct Dir *dir, char *nome, int n, int *achados)
```

- Se não existirem n ocorrências, `busca_dir` retorna 0 e o parâmetro de saída achados diz quantas ocorrências existem
- O caso base agora é `dir->nome` igual ao nome passado, e `n == 1`

Desenhando com recursão

- Desenhar com funções recursivas é uma forma interessante de “visualizar” a recursão, e formar padrões bonitos:

```
static void rec_H(double x, double y, double alt, double larg, int n) {
    tela_ret(x - larg/2, y-1, larg, 2, 1, 1, 1);
    tela_ret(x - larg/2 - 1, y - alt/2, 2, alt, 1, 1, 1);
    tela_ret(x + larg/2 - 1, y - alt/2, 2, alt, 1, 1, 1);
    if(n > 0) {
        rec_H(x - larg/2, y - alt/2, alt/2, larg/2, n-1);
        rec_H(x - larg/2, y + alt/2, alt/2, larg/2, n-1);
        rec_H(x + larg/2, y - alt/2, alt/2, larg/2, n-1);
        rec_H(x + larg/2, y + alt/2, alt/2, larg/2, n-1);
    }
}
```

Concluindo

- Árvores como nossa árvore de diretórios são o tipo mais comum de estrutura recursiva, e aparecem na maioria das aplicações

- Os controles de uma interface gráfica formam uma árvore, uma página HTML em um browser é uma árvore de tags, um programa de computador é uma árvore de declarações, comandos e expressões

- Funções recursivas como as que vimos são a maneira mais fácil de trabalhar com qualquer tipo de árvore, e vocês usarão muito esse tipo de função em outras disciplinas do curso de computação