Universidade Federal do Espírito Santo

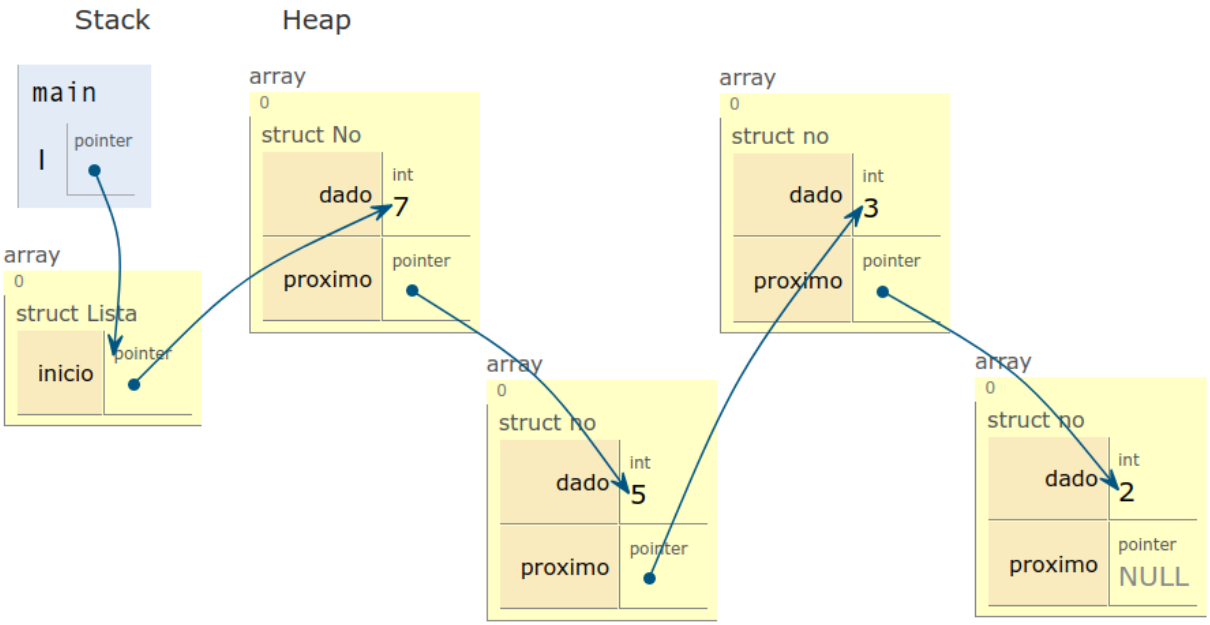
Departamento de Computação e Eletrônica - CEUNES
Estrutura de Dados I
Prof. Oberlan Romão

Listas Simplesmente Encadeadas

Atenção: Ao terminar, não se esqueça de enviar as soluções no AVA.

O objetivo dessa aula prática é nos familiarizarmos com os conceitos de lista simplesmente encadeadas e implementarmos algumas funções comumente presentes em listas. Antes de iniciar, baixe o arquivo `lista.c` no AVA.

Uma das formas mais práticas de entender o funcionamento de listas encadeadas é com "figuras". Abra o site C Tutor e cole o código do arquivo `lista.c`. Clique no botão "`Visualize Execution`". Em seguida, veja o passo-a-passo da execução do programa clicando no botão "`Next >`". Se precisar/quiser, você pode voltar um passo clicando em "`< Prev`". Após a inserção de todos os nós, você deve obter uma figura parecida com a exibida abaixo.



1) Qual a ordem de inserção dos elementos (ou seja, onde os elementos são inseridos na lista)? Qual a vantagem de se fazer dessa forma?

2) Computacionalmente, o que indica que um nó é o último de uma lista encadeada?

3) Note que o campo `proximo` do nó mais a direita (cujo campo `dado` possui valor 2) contem o valor `NULL`. O que isso significa?

4) Analise a função `insere`. Note que não é feito, explicitamente, `no->proximo = NULL` na função, mas (como podemos ver na figura), o último nó contém essa informação (ou seja, `proximo = NULL`).

Um ponteiro, assim como uma variável qualquer, se não for explicitamente inicializada, começa com um valor qualquer (um "lixo"). Onde (ou em que momento) é feito a atribuição de `NULL` ao último nó? Dica: use o C Tutor para te ajudar;

5) Comente a linha 15 do código (`l->inicio = NULL;`). Copile e execute o código novamente. Provavelmente, o código funcionará como esperado.

   a) Qual o objetivo do código da linha 15?

   b) Teste com o `valgrind` para ver se ele encontra algum erro. Ele detectou algum erro?

   c) Volte ao C Tutor, clique em "`Edit this code`" comente a linha 15 e visualize a execução novamente (clicando várias vezes no botão "`Next >`"). O que mudou?

6) Descomente a linha 15 e analise o código da função `imprime`. Ela foi implementada para imprimir um elemento na frente do outro. Considerando as inserções feitas na função `main()` do arquivo `lista.c`, a saída fica:

   `7 5 3 2`

   Modifique a função para que a saída seja da seguinte forma:

   `7->5->3->2->NULL`

7) Baseado nas função `imprimir` e `liberaLista`, implemente uma função que retorna a quantidade de elementos de uma lista (chame-a de `tamanhoLista`).

   a) Qual a complexidade dessa função?

   b) Como podemos modificar o `struct lista` para melhorar a complexidade dessa função?

8) Faça uma função `No *criaNo(int valor)` que aloca e retorna um ponteiro para um nó. O campo `dado` do nó alocado deve receber `valor`, já o campo `proximo` deve receber `NULL`.

9) Considerando a função `criaNo`, faça uma função `void insereNo(Lista *l, No *no)` que recebe um nó alocado e o insere no início da lista. Veja alguns exemplos de utilização dessa função:

```
1 Lista *l = criaLista();
2 ...
3 No *no = criaNo(11);
4 insereNo(l, no);
5 insereNo(l, criaNo(13)); //Note que não precisamos criar uma variável como
                                          feito na linha 3.
6 ...
7 liberaLista(l);
```

10) No trecho de código anterior, precisamos/podemos fazer `free(no)` após o `liberaLista(l)`? Porquê?

11) O `struct no` pode ser visto como uma estrutura recursiva, pois existe um ponteiro para um `struct no`. Com isso, é muito comum implementarmos algumas funções de forma recursiva também. Veja como podemos fazer a função `imprime` de forma recursiva:

```c
void imprimeListaRec(No *no) {
    if(no) {
        printf("%d\n", no->dado);
        imprimeListaRec(no->proximo); // Chama a função recursivamente
                                      passando o próximo nó
    }
}
int main() {
    ...
    imprimeListaRec(l->inicio);
    ...
}
```

Note que para utilizarmos essa função, precisamos passar o ponteiro `inicio`. Posteriormente, iremos modificar essa função.

Na função `imprimeListaRec`, troque a ondem entre o `printf` e a chamada recursiva da função `imprimeListaRec`, ou seja, primeiro chame a função recursivamente e depois imprima. O que mudou?

12) Crie uma função `tamanhoListaRec` que retorna o tamanho de uma lista de forma recursiva.