Atividade prática - Recursão
Professor Wladimir A. Tavares

Recursão com padrões

Nos problemas de recursão com padrões, vamos utilizar um buffer para armazenar os dados antes de exibi-los na tela. Essa abordagem é útil para controlar a saída e manipular os padrões. As funções abaixo implementam um buffer simples para desenhar padrões, como espaços e asteriscos, e incluem funcionalidades básicas de manipulação:

- Inicializar o buffer (`inicia_buffer`);
- Adicionar vetores de caracteres ao buffer (`salva_buffer`);
- Escrever quebras de linha (`pula_linha`);
- Escrever espaços (`print_space`) e asteriscos (`print_asterisco`) de forma recursiva.

O código é apresentado a seguir:

```c
char buffer[1024];
int offset = 0;

void inicia_buffer(){
    memset(buffer, 0, sizeof(buffer));
    offset = 0;
}

void salva_buffer(char * s){
    int r = sprintf(buffer + offset, "%s", s);
    offset += r;
}

void pula_linha(){
    salva_buffer("\n");
}

void print_space(int n){
    if(n == 0) return ;
    else {
        salva_buffer(" ");
        print_space(n-1);
    }
}

void print_asterisco(int n){
    if(n == 0) return ;
    else {
        salva_buffer("*");
        print_asterisco(n-1);
    }
}
```

Triangulo

Implemente uma função recursiva chamada triangulo(n) que gera no buffer o padrão de um triângulo escalonado com n linhas. Por exemplo, para triangulo(5), o padrão escrito no buffer será:

```
*\n
**\n
***\n
****\n
*****\n
```

Essa função deve utilizar o buffer previamente descrito para armazenar o resultado e fazer uso de recursão para construir o padrão linha por linha.

Versão Iterativa

```
void trianguloIterativo(int n){
    for(int i = 1; i <= n; i++){
        print_asterisco(i);
        pula_linha();
    }
}
```

Versão Recursiva

```
void triangulo(int n){
    if(n==1){
        print_asterisco(1);
        pula_linha();
    }else{
        triangulo(n-1);
        print_asterisco(n);
        pula_linha();
    }
}
```

Teste

```
void triangulo_teste(int n){
    char tmp[1024];
    inicia_buffer();
    trianguloIterativo(n);
    strcpy(tmp, buffer);
    inicia_buffer();
    triangulo(n);
    if( strcmp(buffer, tmp) == 0){
        printf("Teste : OK\n");
    }else{
        printf("Teste : FAIL\n");
    }
}
void problema1(){
    printf("Problema 1\n");
    triangulo_teste(10);
}
```

A função `triangulo_teste(int n)` foi desenvolvida para validar a correção da implementação recursiva da função triangulo(n) em comparação com uma implementação iterativa equivalente, chamada `trianguloIterativo(n)`. O objetivo é garantir que ambas as abordagens produzam exatamente o mesmo resultado ao gerar o padrão de triângulo no buffer.

Exercícios

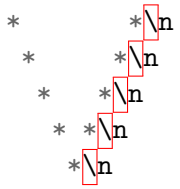
1. Implemente a versão recursiva da função `retangulo(n, m)`. Essa função deve escrever no buffer um retângulo composto por n linhas e m colunas de asteriscos. Cada linha é seguida por uma quebra de linha ($\backslash n$).

   Por exemplo, para retangulo(4, 5), o padrão escrito no buffer será:

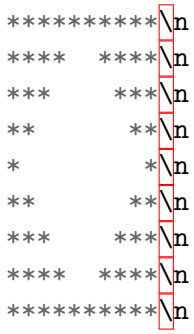
2. Implemente uma versão da função `letraV(int n)`. Essa função deve escrever no buffer uma letraV composta por n linhas.

   Por exemplo, para letraV(5), o padrão escrito no buffer será:

3. Implemente uma versão recursiva da função `bandeira(int n)`. Essa função deve escrever no buffer uma bandeira composta por 2n linhas.

   Por exemplo, para bandeira(5), o padrão escrito no buffer será:

4. Implemente uma versão recursiva da função `ordenado(int v[], int n)`. Essa função deve verificar se um vetor de tamanho n está ordenado em ordem crescente.

   Dica: Para resolver o problema recursivamente, considere o seguinte:

   Se sabemos que a sublista v[0..n-2] está ordenada, como podemos verificar que a lista completa v[0..n-1] também está ordenada?

   Pense em como dividir o problema em subproblemas menores até atingir o caso base.

5. Implemente uma versão recursiva da função `inverte(char *s)` que inverte um vetor de caracteres.

   Dica: Para resolver este problema, utilizaremos uma função auxiliar com três parâmetros: o vetor de caracteres s, o índice inicial i e o índice final j. A ideia é trocar os caracteres nas posições i e j, reduzindo recursivamente o intervalo até que os índices se encontrem ou se cruzem.

   O protótipo da função auxiliar é:

   ```c
   void __inverte(char * s, int i, int j){
       return ;
   }
   ```

   A função principal chama a função auxiliar, definindo os índices iniciais:

```c
void inverte(char * s){
    int n = strlen(s);
    __inverte(s, 0, n-1);
}
```