Introdução à Programação C
CMT012
Aula 14

Ronald Souza
Instituto de Computação - UFRJ
ronaldsouza@dcc.ufrj.br

Conteúdo de hoje

Ponteiros - parte 3/3 - alocação dinâmica

Arquivos - parte 3/3 - mais funções e operações

Alocação dinâmica de memória

→ Ponteiros permitem alocar e desalocar **lotes** da memória *heap* durante a execução do programa (assim como vetores alocam lotes da *stack*).

→ O programador pode alocar o número exato de posições que o programa requer.
Para isso, usar funções da biblioteca **stdlib.h**:
***Pedem** ao **sistema operacional** para alocar **lotes** da memória (heap) e devolvem ao programa o endereço inicial deste local.*

Funções para alocação dinâmica

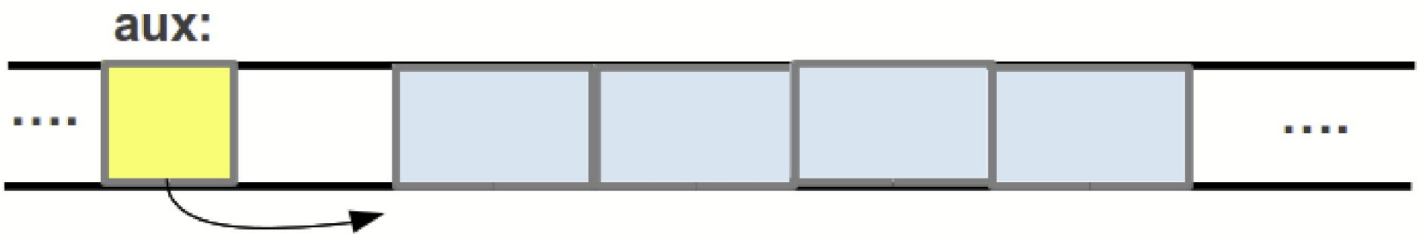
void *malloc(size t qtde): reserva qtde bytes de espaço na memória.
Retorna um ponteiro para o espaço reservado ou NULL no caso de algum erro ocorrer.

void *calloc(size t num, size t tamanho): reserva espaço na memória para um vetor de num itens, cada item tem tamanho **tamanho** (em bytes) e todos os bits do espaço são inicializados com 0.
Retorna um ponteiro para o espaço reservado ou NULL no caso de algum erro ocorrer.

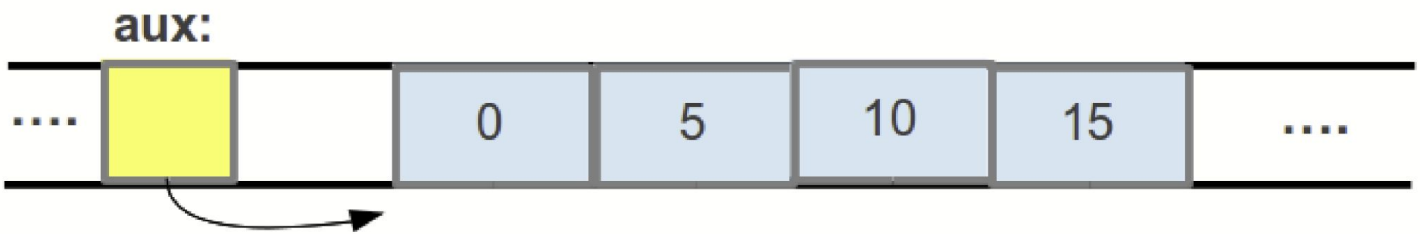
void free(void *pont): libera o espaço de memória apontado por pont.

malloc(): exemplo


```
int *aux, i;
aux = malloc (4 * sizeof(int));
if (aux != NULL) ...
```




```
...
for(i=0; i<4; i++)
   *(aux+i) = i * 5;
```



calloc() e free(): exemplo

```
#include<stdio.h> #include<stdlib.h>
int main () {
    float *v; int i, tam;
    printf ("Qual o tamanho do vetor? ");
    scanf ("%d" , &tam);
    v = (float*) calloc(tam, sizeof(float));
    if (!v) {printf ("Erro ao alocar memoria"); return -1;}

    for (i=0; i<tam; i++) {
        printf ("Elemento %d? " , i) ;
        scanf ("%f", v+i);
    }
    for (i=0; i<tam; i++) printf ("v[%d]=%f\n",i,*(v+i));
    free(v); return 0;
}
```

Leitura e escrita de strings: fgets() e fputs()

As funções **fgets()** e **fputs()** servem para ler e escrever cadeias de caracteres em arquivos

→ Protótipos:

```
int   fputs(char *str, FILE *parq);
char* fgets(char *str, int comp, FILE *parq);
```

Leitura e escrita de strings: fgets() e fputs()

```
int   fputs(char *str, FILE *parq);
char* fgets(char *str, int comp, FILE *parq);
```

→ fputs(): escreve a string apontada por **str** no arquivo

→ fgets(): lê uma string do arquivo até um caractere de **nova linha** ou até **comp - 1** (acrescenta o caractere **nulo** ao final). *Retorna o ponteiro para a cadeia de caracteres lido ou NULL se ocorrer erro ou* ***final de arquivo****.*

fgets(): exemplo

→ **exemplo.txt:**

```
primeira linha
segunda linha
```

→**Código:**

```
char str1 [50]; char str2 [40];
FILE* arquivo;
if (!(arquivo = fopen ("exemplo.txt", "r"))) {
    printf (" Erro na abertura de arquivo!\n");
    exit(1);
}
fgets (str1, 50, arquivo);
//fscanf(arquivo, "%s", str);          //daria no mesmo?
//fscanf(arquivo, "%[^\n]", str);      //e esse, daria no mesmo?
fgets (str2, 40, arquivo);
```

fputs(): exemplo

```
char *s;
FILE *arq;
if ((s = (char*) malloc(100)) == NULL)
    return -1;

arq = fopen("exemplo.txt", "r");

if(!arq)
    return -1;

while((s = fgets(s, 100, arq)))
    fputs(s, stdout); //saída padrão!

fclose(arq);
free(s); //Libera da memória o espaço alocado por s
return 0;
```

Leitura e escrita de caracteres: fgetc() e fputc()

```
int fgetc(FILE *parq);
int fputc(int ch, FILE *parq);
```

⟶ As operações mais simples em arquivos são **leitura e escrita de caracteres**.

⟶ fgetc(): retorna **EOF** (*flag* de **final do arquivo**) quando não restam mais caracteres a serem lidos no arquivo **a partir da posição atual (mais detalhes a seguir)**.

fgetc() e fputc(): exemplo

```
int c;
FILE *arq;
arq = fopen("exemplo.txt", "r");

if(arq == NULL) //O mesmo que “if(!arq)”
    return -1;

while((c = fgetc(arq)) != EOF)
    fputc(c, stdout);

fclose(arq);
return 0;
```

Posicionamento dentro do arquivo

→ Quando um arquivo é aberto, um **indicador de posição** é associado a ele. O indicador determina o ponto do arquivo a partir do qual executar a próxima instrução de leitura ou escrita.

→ Pode ser imaginado como o cursor de um editor de texto

Posicionamento dentro do arquivo: Final do Arquivo - EOF

→ A função **feof()** indica que um arquivo chegou ao seu final:

```
int feof(FILE *parq)
```

→ Seu uso é importante em arquivos binários (onde EOF pode ser lido como parte do arquivo)

→ Retorna 0 se ainda não chegou no final do arquivo, e valor diferente de zero caso contrário

OBS: EOF = *End of File (i.e. "fim do arquivo")

feof(): exemplo

```
int c;
FILE *arq;
arq = fopen("exemplo.txt", "r");

if(arq == NULL) //O mesmo que “if(!arq)”
    return -1;
c = fgetc(arq);
while(!feof(arq)) {
    fputc(c, stdout);
    c = fgetc(arq);
}
fclose(arq);
return 0;
```

Posicionamento dentro do arquivo: Voltar ao início do arquivo - rewind()

→ A função **rewind()** reposiciona o indicador para o início do arquivo:

```
void rewind(FILE *parq);
//Ex.: escreve e em seguida lê o conteúdo do arquivo
```

Posicionamento dentro do arquivo: Reposicionando-se — fseek()

→ É possível reposicionar o indicador de posição no arquivo para **qualquer posição** entre o **início** e **EOF**, com a função **fseek().**

→ Muito util, por ex., quando se sabe o índice de uma estrutura específica já gravada, dentre as estruturas gravadas em um arquivo binário. Pode-se ler essa estrutura **sem carregar-se para a memória todas as demais!**

Posicionamento dentro do arquivo: Reposicionando-se — fseek()

```
int fseek (FILE* fp, long numbytes, int origin);
```

→ Retorna 0 se sucesso e **distância em bytes p/ o início do arquivo** caso contrário

fp: apontador para arquivo já aberto via fopen()

numbytes: de quantos bytes deslocar o indicador **a partir de *origin***

origin: referência de reposicionamento. Será uma das três macros:

- SEEK SET: início do arquivo
- SEEK CUR: posição atual
- SEEK END: final do arquivo (EOF)

Exemplo:

```
fseek(arq, 2*sizeof(int), SEEK_SET);
```
//3o. inteiro a partir do INÍCIO.

fseek(): exemplo

```
#include<stdio.h>
int main () {
    int c;
    FILE *arq;
    arq = fopen("exemplo.txt", "r");
    if(arq == NULL)
        return -1;
    fseek(arq, -20, SEEK_END);
    while((c = fgetc(arq)) != EOF) {
        fputc(c, stdout);
    }
    fclose(arq);
    return 0;
}
```

Por hoje é isso!

Slides baseados no material de Silvana Rossetto.