Aula 05

Algoritmos e Programação II

Setembro / 2017

Prof. Mario Liziér
lizier@ufscar.br

Alocação Estática vs Dinâmica

- Até o momento, quando queremos armazenar algo na memória, declamos variáveis e o respectivo espaço é alocado e desalocado automaticamente!
- As variáveis “existem” enquanto estivermos com seu escopo “ativo”
  - Assim que um escopo termina, todas as variáveis são “des-declaradas” e o seu espaço é então desalocado
- Este modelo é chamado de alocação estática

- Com alocação dinâmica de memória, podemos controlar a alocação e desalocação de espaços (mas não podemos controlar a declaração e “des-declaração” de variáveis!)

Alocação Dinâmica

- Podemos então solicitar ao sistema espaço de memória (normalmente em *bytes*) e utilizarmos o tempo que quisermos, independente de escopo!
  - Em compensação, precisamos sempre:
    - Manter **sob nosso controle** o endereço do espaço alocado dinâmicamente (se perdermos onde o espaço foi alocado, não conseguimos mais acessá-lo)
    - Solicitar a **desalocação** quando não quisermos mais utilizar (liberando aquele espaço para novas solicitações)

Alocação Dinâmica

- Na <stdlib.h> temos algumas funções para alocar memória dinâmicamente:

  malloc – Aloca um bloco de memória mas não o inicializa (função mais utilizada)

  ```
  void *malloc(size_t size);
  ```

  calloc - Aloca um bloco de memória e o inicializa (menos eficiente que malloc);

  ```
  void *calloc(size_t nmemb, size_t size);
  ```

  realloc – Redimensiona um bloco de memória previamente alocado.

  ```
  void *realloc(void *ptr, size_t size);
  ```

Alocação Dinâmica

- malloc
  - Retorna um (void*), ou seja, um endereço de memória sem tipo!
    - Comum fazermos um *cast* no retorno, convertendo o (void*) para o tipo correto, por exemplo: (int*) ou (float*) ou (double*) ou ...
  - Recebe como parâmetro o número de *bytes* do bloco que memória a ser alocado
    - Comum o uso do operador *sizeof* , para obtermos dinamicamente o número de *bytes* do tipo que será utilizado
  - Uso recomendado:

```
float *p = (float*)malloc(N*sizeof(float));
char *p = (char*)malloc(N*sizeof(char));
int *p = (int*)malloc(N*sizeof(int));
double *p = (double*)malloc(N*sizeof(double));
```

Alocação Dinâmica

- malloc
  - Se a função malloc não conseguir encontrar espaço contíguo suficiente, irá retornar NULL
    - É importante verificar o retono antes de acessar o conteúdo do endereço retornardo!
- free
  - Para desalocar o espaço solicitado, devemos utilizar o subprograma *free*

```
void free(void*);
```

Alocação Dinâmica

- Exemplo:

```c
/* malloc example: random string generator*/
#include <stdio.h>      /* printf, scanf, NULL */
#include <stdlib.h>     /* malloc, free, rand */

int main ()
{
  int i,n;
  char * buffer;

  printf ("How long do you want the string? ");
  scanf ("%d", &i);

  buffer = (char*) malloc (i+1);
  if (buffer==NULL) exit (1);

  for (n=0; n<i; n++)
    buffer[n]=rand()%26+'a';
  buffer[i]='\0';

  printf ("Random string: %s\n",buffer);
  free (buffer);

  return 0;
}
```

Alocação Dinâmica

- Exemplo 2:

```c
#include <stdio.h>
#include <stdlib.h>

int main(void){
    double *n; /* ponteiro para o espaço a ser alocado */
    n = (double *) malloc(sizeof(double));
    if (!n){ /* testa a alocação */
        printf("Não conseguiu alocar a memória\n");
        exit(1);
    }
    /* usa o double apontado por n */
    ...
    /* libera a memória alocada */
    free(n);
    /* o programa continua, n contém agora um indereço inválido */
    ...
    return (0);
}
```

Alocação Dinâmica

- Cuidados:
  - Temos que ter sempre um ponteiro apontando para o bloco de memória alocado dinamicamente
  - Entre escopos diferentes, precisamos muitas vezes passar o ponteiro como parâmetro de entrada ou saída/retorno
  - (**vazamento de memória**) Se esquecermos de chamar o procedimento free, ou então perdemos o(s) ponteiro(s) que apontava(m) para o bloco de memória alocado dinamicamente, não conseguiremos mais
    - Acessar
    - Chamar o procedimento free

Alocação Dinâmica

- **calloc**
  - Retorna um (void*), ou seja, o endereço de memória alocado dinamicamente, mas sem tipo
  - Retona NULL caso não encontre espaço suficiente
  - Recebe dois parâmetros: o número de elementos e o tamanho em *bytes* de cada elemento
    - Já fáz a multiplicação que precisamos fazer no malloc
- **realloc**
  - Retorna um (void*), ou seja, o endereço de memória realocado dinamicamente e se houve mudança, o endereço antigo agora é inválido, pois foi desalocado
  - Retona NULL caso não encontre espaço suficiente, e neste caso, o espaço antigo não é alterado
  - Recebe dois parâmetros: o endereço do espaço a ser realocado e o tamanho em *bytes* do novo bloco

Exercício

- Faça um programa que gere X pontos aleatórios no espaço
  - Aloque o vetor de pontos dinâmicamente