Alocação dinâmica de memória

Conceitos

- Certas aplicações demandam uma área de armazenamento de valores que não pode ser estimada em tempo de compilação;
- Eventualmente, existe a necessidade de se alocar (criar) variáveis dinamicamente (em tempo de execução);
- Tais variáveis são alocadas numa área denominada HEAP;
- O HEAP é gerenciado pelo sistema de execução;
- A aplicação (programa C) pode requisitar a alocação de uma área no HEAP ou então liberar a mesma para outro uso;
- O tempo de vida dessas variáveis é controlado de forma direta pelo programador;
- Para isso, são usadas variáveis do tipo ponteiro.

Programa

Variáveis globais

Pilha (variáveis locais)

Funções

- São usadas as funções `malloc()` e `free()` da biblioteca `stdlib.h`
- A primeira serve para alocar um espaço de memória do HEAP; a quantidade de bytes requisitados é passada como parâmetro; o ponteiro para a área alocada é devolvido;
- A segunda serve para liberar a área de memória ocupada por uma variável (a quantidade de bytes está implícita).

malloc()

```
void *malloc (int size)
```

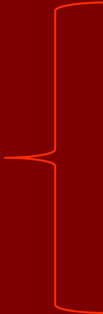
- `size` denota a quantidade de bytes requisitados (área contígua)
- `void *` significa que o endereço retornado pode ser atribuído para qualquer variável do tipo ponteiro.

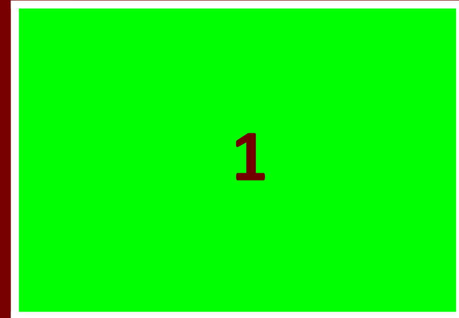
```
int *p;
p=malloc(4); // ou malloc(sizeof(int))
*p=1;
```

p

2000

4 bytes





*p

malloc()

```
void *malloc (int size)
```

- se não houver espaço suficiente no HEAP para atender a solicitação, `malloc` retorna NULL;

```
int *p;
p=malloc(4);
if (p) ... // alocação bem-sucedida
```

free()

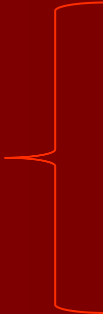
```
void free (void *p)
```

- libera uma área de memória no HEAP com tamanho inferido a partir do tipo de dados apontado pelo argumento passado para a função;
- é necessário que o ponteiro seja válido;
- o argumento permanece inalterado no retorno e deve-se evitar o uso do mesmo após a chamada da função.

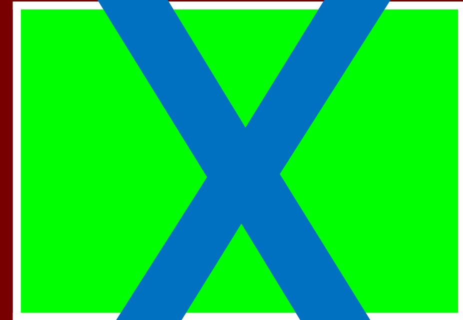
```
int *p;
p=malloc(4);
...
free(p);
```

p

2000



4 bytes



*p

Precauções

- Não usar um ponteiro que não tenha sido inicializado previamente (caso contrário os resultados são imprevisíveis);
- Requsitar a quantidade de bytes necessária no `malloc` (nem mais nem menos);
- Não esquecer de liberar áreas que não serão mais utilizadas (`free`);
- Não usar ponteiros para áreas previamente liberadas (caso contrário os resultados são imprevisíveis);
- Não atribuir o endereço de uma área recém-alocada para um ponteiro que ainda se refere (apenas ele) a uma outra área do HEAP;
- O uso intensivo de `malloc` e `free` pode segmentar o HEAP de tal forma que eventuais novas solicitações de área podem não ser mais atendidas.

Exemplos

```
int main () {
   int *p;
   *p=0; // ponteiro inválido!
   ...
```

Exemplos

```
int main () {
  int *p;
  p=malloc(sizeof (int));
  p=malloc(sizeof (int));
  // a área anterior torna-se inacessível
  ...
```

Exemplos

```
int main () {
  int *p;
  p=malloc(5); // área maior
  p=malloc(3); // área menor
  *p=0; // invasão de área!
  ...
```

Exemplos

```
int main () {
  int *p,*q;
  p=malloc(sizeof (int));
  q=p;
  free (p);
  *q=0 // ponteiro inválido!
  ...
```