Notas de aula 28/03

Alocação dinâmica de memória

O compilador reserva espaço na memória para todos os dados declarados explicitamente:

```c
int x; // reserva 4 bytes na memória
char c; // reserva 1 byte na memória
int v[5]; // reserva 5 x 4 bytes na memória;
char s[10]; // reserva 10 x 1 byte na memória;
```

No caso de variáveis do tipo ponteiro, precisamos vincular o ponteiro a um endereço de memória existente. Podemos, por exemplo, vincular um ponteiro do tipo inteiro ao endereço de uma variável do mesmo tipo:

```c
int *p, x;
p = &x; // p aponta para o endereço de memória de x.
```

E se em vez de vincular o ponteiro p à um endereço que já existe, eu quiser vincular p a um novo endereço de memória? Ou seja, um endereço que não está previamente alocado? Para isso teríamos que alocar espaço em memória durante a execução do programa. O processo de alocar memória durante a execução é chamado de **alocação dinâmica de memória**.

A biblioteca stdlib.h possui funções úteis para trabalhar com alocação dinâmica de memória.

Função malloc

O nome dessa função vem de *memory allocation*. Podemos deduzir então, que esta função tem a responsabilidade de alocar uma quantidade de memória. Ela recebe como parâmetro a quantidade de memória que deverá ser alocada (em bytes) e retorna um ponteiro para o primeiro endereço alocado.

Por exemplo: se quisermos alocar memória suficiente para alocar um dado do tipo inteiro, precisamos alocar 4 endereços de memória:

```c
int *p = malloc(4); // p aponta para um novo endereço
```

Para facilitar podemos usar o sizeof:

```c
int *p = malloc(sizeof(int)); // p aponta para um novo endereço
```

Se quisermos alocar memória para um vetor que guarda 5 números inteiros, podemos fazer:

```c
int *p = malloc(5 * sizeof(int));
```

Nesse caso, 20 endereços de memória são alocados e p aponta para o primeiro desses endereços.

Considere agora uma estrutura:

```
struct ponto {
      float x;
      float y;
};
typedef struct ponto Ponto;
```

Para alocar memória para um item do tipo Ponto, faríamos:

```
Ponto *p = malloc(sizeof(Ponto));
```

Ou se desejássemos alocar memória para um vetor com 10 itens do tipo Ponto:

```
Ponto *p = malloc(10 * sizeof(Ponto));
```

O tipo Ponto ocupa 8 bytes, pois é formado por dois tipo float. Desta forma, um vetor com 10 elementos do tipo Ponto irá ocupar 80 bytes na memória. Portanto, ao chamar a função malloc, 80 endereços de memória serão alocados e o ponteiro p apontará para o primeiro desses endereços.

Função calloc

A função malloc tem o objetivo de alocar endereços memória, mas não se preocupa com o conteúdo desses endereços. Pode ser que os endereços alocados contenham algum lixo de memória. Em contrapartida, a função calloc, além de alocar endereços de memória, garante que o conteúdo desses endereços é 0. Ela espera como parâmetro o número de elementos que precisa ser alocado e o tamanho de cada elemento:

```
int *p = calloc(1, sizeof(int)); // aloca endereço para 1 int
int *v = calloc(10, sizeof(int)); // aloca endereço para 10 int
```

Função free

As variáveis alocadas estaticamente são eliminadas após a execução do programa. O mesmo não acontece com variáveis alocadas dinamicamente. Elas continuam existindo mesmo depois que a execução termina. Por isso, após usar essas variáveis é preciso liberar manualmente a memória ocupada por elas. Por exemplo:

```
int *p = malloc(5 * sizeof(int));

// processamento

free(p);
```

Função realloc

Permite redimensionar o bloco de memória apontado por um ponteiro, tanto para mais, quanto para menos. Ela recebe como parâmetro um ponteiro e o novo tamanho do bloco, em bytes. Seu retorno é um ponteiro para o primeiro endereço do novo bloco de memória. Exemplo:

```
char *pc = malloc(30 * sizeof(char));
pc = realloc(pc, 20 * sizeof(char));
```