MC102 - Algoritmos e Progração de Computador

Prof. Alexandre Xavier Falcão

20° Aula: Alocação dinâmica de memória

1 Declarando e manipulando apontadores

Antes de falarmos de alocação dinâmica de memória, vamos entender como variáveis do tipo apontador são definidas e como elas podem ser manipuladas em memória.

```
int main()
{
  int v[10],*u;

  v[0] = 30;
  printf("%d\n",*v); /* imprime o conteúdo apontado por v, que é v[0] */
  u = v; /* atribui o endereço do vetor a u */
  printf("%d\n",u);
  u[4] = 10; /* atribui 10 a u[4], que é v[4] */
  printf("%d\n",v[4]);
  u = &v[2]; /* atribui o endereço de v[2] a u */
  printf("%d\n",u);
  u[2] = 20; /* atribui 20 a u[2], que é v[4] */
  printf("%d\n",v[4]);
}
```

A memória disponível para um programa é dividida em três partes. Uma parte é reservada para o código executável do programa, e as outras duas são reservadas para variáveis definidas durante a execução do programa. Por exemplo, o programa acima utiliza o espaço disponível na **pilha**— parte da memória reservada para variáveis locais, incluindo as variáveis associadas a funções— para armazenar as variáveis v, u, v[0], v[1], ..., v[9]. Este espaço é normalmente insuficiente para armazenar muitas variáveis, como é o caso de vetores muito grandes. Uma outra situação ocorre quando não sabemos *a priori* o tamanho do vetor. Para esses casos, existe uma parte para **dados** na memória, bem maior que a pilha, a qual pode ser alocada pelo programador. Nesta aula, vamos aprender como definir e manipular esses espaços em memória.

2 Alocando memória para vetores

O programa abaixo mostra como alocar e desalocar espaço em memória durante a sua execução.

```
#include <stdio.h>
#include <malloc.h> /* biblioteca de funções de alocação de memória */

typedef struct _ponto {
  float x,y;
} Ponto;

int main()
{
  Ponto *p=NULL; /* variável do tipo apontador para Ponto */
  int *v=NULL; /* variável do tipo apontador para int */
  char *text=NULL; /* variável do tipo apontador para char */
  int n;

  scanf("%d",&n);
  v = (int *)calloc(n,sizeof(int)); /* aloca n elementos inteiros em memória
                                       e retorna o endereço do primeiro, ou
                                       NULL caso não tenha sido alocada. */
  scanf("%d",&n);
  text = (char *)calloc(n,sizeof(char)); /* aloca espaço para cadeia
                                            com n caracteres. */
  scanf("%d",&n);
  p = (Ponto *)calloc(n,sizeof(Ponto)); /* aloca espaço para vetor de
   pontos com n pontos. */

  printf("v %d text %d p %d\n",v,text,p); /* imprime os endereços */

  v[0] = 10;
  sprintf(text,"Oi"); /* text = "Oi" está errado neste caso, pois o
  computador alocará novo espaço para armazenar a cadeia e atribuirá o
  endereço deste espaço para text. Portanto, o endereço alocado
  inicialmente para text ficará perdido. Assim, devemos usar o comando
  sprintf para gravar a cadeia no espaço alocado para text. */

  p[0].x = 20;
  p[0].y = 30;
  printf("%d\n",v[0]);
  printf("%s\n",text);
  printf("(%f,%f)\n",p[0].x,p[0].y);

  if (v != NULL) {
    free(v); /* libera memória */
    v = NULL;
  }

  if (text != NULL) {
```

```
    free(text); /* libera memória */
    text = NULL;
  }

  if (p != NULL) {
    free(p); /* libera memória */
    p = NULL;
  }
return 0;
}
```

Outra opção para alocação de memória é usar a função *malloc*:

```
v = (int *) malloc(n*sizeof(int));      /* só possui 1 argumento */
text = (char *) malloc(n*sizeof(char)); /* só possui 1 argumento */
p = (Ponto *) malloc(n*sizeof(Ponto));  /* só possui 1 argumento */
```

A única diferença entre *malloc* e *calloc*, fora o número de argumentos, é que *calloc* inicializa a memória com zeros/espaços em branco.

Portanto, podemos usar a sintaxe abaixo para alocar espaço para vetores de todos os tipos conhecidos.

```
tipo *var=(tipo *) calloc(número de elementos,sizeof(tipo));
tipo *var=(tipo *) malloc(número de elementos*sizeof(tipo));
```

3 Alocando memória para estruturas abstratas

Por uma questão de organização, quando usamos registro para definir uma estrutura abstrata, nós devemos definir uma função para cada tipo de operação com esta estrutura. Por exemplo, suponha uma estrutura para representar uma figura formada por um certo número de pontos interligados por segmentos de reta. As operações desejadas são alocar memória para a figura, ler os pontos da figura, imprimir os pontos lidos, e desalocar memória para a figura. Estas operações definem, portanto, quatro funções conforme o programa abaixo.

```
#include <stdio.h>
#include <malloc.h>

typedef struct _ponto {
  float x,y; /* coordenadas Cartesianas de um ponto */
} Ponto;

typedef struct _figura {
  Ponto *p; /* pontos da figura */
  int n;    /* número de pontos */
} Figura;

Figura CriaFigura(int n); /* aloca espaço para figura com n pontos e
```

```
                              retorna cópia do registro da figura */
void   DestroiFigura(Figura f); /* desaloca espaço */
Figura LeFigura(); /* lê o número de pontos, cria a figura, ler os
                      pontos, e retorna cópia do registro da
                      figura. */
void   ImprimeFigura(Figura f); /* imprime os pontos */

Figura CriaFigura(int n)
{
  Figura f;

  f.n = n;
  f.p = (Ponto *) calloc(f.n,sizeof(Ponto));
  return(f); /* retorna cópia do conteúdo de f */
}

void DestroiFigura(Figura f)
{
  if (f.p != NULL) {
    free(f.p);
  }
}

Figura LeFigura()
{
  int i,n;
  Figura f;

  scanf("%d",&n); /* lê o número de pontos */
  f = CriaFigura(n); /* aloca espaço */
  for (i=0; i < f.n; i++) /* lê os pontos */
    scanf("%f %f",&(f.p[i].x),&(f.p[i].y));
  return(f); /* retorna cópia do conteúdo de f */
}

void ImprimeFigura(Figura f)
{
  int i;

  for (i=0; i < f.n; i++)
    printf("(%f,%f)\n",f.p[i].x,f.p[i].y);
}

int main()
{
  Figura f;
```

```
  f=LeFigura();
  ImprimeFigura(f);
  DestroiFigura(f);
  return(0);
}
```

Um defeito do programa acima é que todos os campos do registro f são armazenados na pilha. No caso de registros com vários campos, estamos disperdiçando espaço na pilha. Podemos evitar isto se definirmos uma **variável apontador para o tipo Figura**, em vez de uma variável do tipo Figura. Esta mudança requer as seguintes alterações no programa.

```
#include <stdio.h>
#include <malloc.h>

typedef struct _ponto {
  float x,y; /* coordenadas Cartesianas de um ponto */
} Ponto;

typedef struct _figura {
  Ponto *p; /* pontos da figura */
  int n;    /* número de pontos */
} Figura;

Figura *CriaFigura(int n); /* Aloca espaço para figura com n pontos e
      retorna endereço do registro da figura */
void   DestroiFigura(Figura **f); /* Destrói espaço alocado e atribui
                                     NULL para a variável apontador
                                     f. Paratanto, precisamos passar o
                                     endereço do apontador, que é uma
                                     variável do tipo apontador de
                                     apontador, ou melhor, apontador
                                     duplo. */
Figura *LeFigura(); /* Lê o número de pontos, cria figura, lê os
                       pontos, e retorna o endereço do registro da
                       figura */
void   ImprimeFigura(Figura *f); /* Imprime os pontos */

Figura *CriaFigura(int n)
{
  Figura *f=(Figura *)calloc(1,sizeof(Figura)); /* aloca espaço para 1
                                                   registro do tipo
                                                   Figura e retorna
                                                   seu endereço em f */
```

```
  f->p = (Ponto *) calloc(n,sizeof(Ponto)); /* aloca espaço para vetor
                                               de pontos e retorna seu
                                               endereço no campo f->p
                                               do registro apontado
                                               por f. Note a mudança
                                               de notação de f.p para
                                               f->p, pois f agora é
                                               apontador de registro e
                                               não mais registro */

  f->n = n; /* copia o número de pontos para o campo f->n */

  return(f); /* retorna endereço do registro alocado em memória */
}

void DestroiFigura(Figura **f)
{
  if (*f != NULL) { /* *f representa o conteúdo do apontador duplo,
                       que é o endereço armazenado no apontador f. */
    if ((*f)->p != NULL)
      free((*f)->p);
    *f = NULL; /* atribui NULL para a variável do escopo principal. */
  }
}

Figura *LeFigura()
{
  int i,n;
  Figura *f=NULL;

  scanf("%d",&n);
  f = CriaFigura(n);
  for (i=0; i < f->n; i++)
    scanf("%f %f",&(f->p[i].x),&(f->p[i].y));
  return(f); /* retorna endereço do registro alocado em memória */
}

void ImprimeFigura(Figura *f)
{
  int i;

  for (i=0; i < f->n; i++)
    printf("(%f,%f)\n",f->p[i].x,f->p[i].y);
}

int main()
```

```
{
  Figura *f=NULL; /* variável do tipo apontador para registro */

  f=LeFigura();
  ImprimeFigura(f);
  DestroiFigura(&f); /* passa f por referência, i.e. o endereço de f. */
  return(0);
}
```