MC202 - Estruturas de Dados

Alexandre Xavier Falcão

Instituto de Computação - UNICAMP

afalcao@ic.unicamp.br

Registros e tipos abstratos de dados

- Um registro é uma variável composta heterogênea e, portanto, pode ser usada para definir qualquer tipo abstrato de dados: ponto, polígono, imagem, pilha, lista, árvore, grafo, etc.

Registros e tipos abstratos de dados

- Um registro é uma variável composta heterogênea e, portanto, pode ser usada para definir qualquer tipo abstrato de dados: ponto, polígono, imagem, pilha, lista, árvore, grafo, etc.

- No caso de sequências numéricas, por exemplo, uma variável do tipo abstrato Vetor (Sequencia) pode conter os valores da sequência e o número de elementos.

Registros e tipos abstratos de dados

- Um registro é uma variável composta heterogênea e, portanto, pode ser usada para definir qualquer tipo abstrato de dados: ponto, polígono, imagem, pilha, lista, árvore, grafo, etc.

- No caso de sequências numéricas, por exemplo, uma variável do tipo abstrato Vetor (Sequencia) pode conter os valores da sequência e o número de elementos.

```
typedef struct _vetor {
  float *val; // vetor de reais
  int nelems; // número de elementos
} Vetor;
```

Registros e tipos abstratos de dados

- Um registro é uma variável composta heterogênea e, portanto, pode ser usada para definir qualquer tipo abstrato de dados: ponto, polígono, imagem, pilha, lista, árvore, grafo, etc.

- No caso de sequências numéricas, por exemplo, uma variável do tipo abstrato Vetor (Sequencia) pode conter os valores da sequência e o número de elementos.

```
typedef struct _vetor {
  float *val; // vetor de reais
  int nelems; // número de elementos
} Vetor;
```

Isso elimina a necessidade de passar o apontador para o vetor e o seu tamanho como dois argumentos de entrada nas funções.

Agenda

Tipos abstratos de dados:

- definição,
- alocação de memória,
- manipulação, e
- liberação de memória no *heap*.

Tipo Vetor

```
typedef struct _vetor {
  float *val; // vetor de reais
  int nelems; // número de elementos
} Vetor;
```

Podemos declarar uma variável do tipo Vetor.

int nelems = 5;
Vetor v = CriaVetor(nelems);

Tipo Vetor

```
typedef struct _vetor {
  float *val; // vetor de reais
  int nelems; // número de elementos
} Vetor;
```

Podemos declarar uma variável do tipo Vetor.

int nelems = 5;
Vetor v = CriaVetor(nelems);

ou um ponteiro para uma variável do tipo Vetor.

int nelems = 5;
Vetor *u = CriaVetor(nelems);

Tipo Vetor

O segundo caso é um pouco mais complexo de manipular, mas traz diversas vantagens:

Tipo Vetor

O segundo caso é um pouco mais complexo de manipular, mas traz diversas vantagens:

- economiza memória na pilha para registros com muitos campos,

Tipo Vetor

O segundo caso é um pouco mais complexo de manipular, mas traz diversas vantagens:

- economiza memória na pilha para registros com muitos campos,
- facilita o empacotamento da função em Python via swig,

Tipo Vetor

O segundo caso é um pouco mais complexo de manipular, mas traz diversas vantagens:

- economiza memória na pilha para registros com muitos campos,
- facilita o empacotamento da função em Python via swig,
- facilita a alocação de vetores e matrizes cujos elementos são do tipo abstrato (e.g., vetores de imagens), e

Tipo Vetor

O segundo caso é um pouco mais complexo de manipular, mas traz diversas vantagens:

- economiza memória na pilha para registros com muitos campos,
- facilita o empacotamento da função em Python via swig,
- facilita a alocação de vetores e matrizes cujos elementos são do tipo abstrato (e.g., vetores de imagens), e
- permite a modificação direta do conteúdo dos campos nas funções.

Tipo Vetor

Os dois casos podem ser representados pelas seguintes funções.

```c
Vetor CriaVetor1(int nelems)
{
  Vetor v;

  v.val = (float *)calloc(nelems,sizeof(float));
  v.nelems = nelems;

  return(v);
}

void DestroiVetor1(Vetor *v)
{
  if (v->val != NULL){
    free(v->val);
    v->val = NULL; v->nelems = 0;
  }
}
```

```c
Vetor *CriaVetor2(int nelems)
{
  Vetor *u = (Vetor *)calloc(1,sizeof(Vetor));

  u->val = (float *)calloc(nelems,sizeof(float));
  u->nelems = nelems;

  return(u);
}

void DestroiVetor2(Vetor **u)
{
  if ((*u) != NULL){
    if ((*u)->val != NULL)
      free((*u)->val);
    free(*u);
    (*u) = NULL;
  }
}
```

Vamos entender como isso funciona no `www.pythontutor.com` com o programa registros.c.

Exercício com o caso 1

Escreva uma função que translada de (Dx,Dy) a posição de um círculo.

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

typedef struct _circulo {
  Ponto centro;
  float raio;
} Circulo;
```

Exercício com o caso 1

```
void TransladaCirculo(Circulo *C, float Dx, float Dy)
{
    (*C).centro.x += Dx;
    (*C).centro.y += Dy;
}
```

Exercício com o caso 1

```
void TransladaCirculo(Circulo *C, float Dx, float Dy)
{
    (*C).centro.x += Dx;
    (*C).centro.y += Dy;
}
```

Como "C" é ponteiro, podemos substituir "(*C)." por "C→" para acessar o conteúdo apontado por "C".

```
void TransladaCirculo(Circulo *C, float Dx, float Dy)
{
    C→centro.x += Dx;
    C→centro.y += Dy;
}
```

Exercício com o caso 1

Um **erro grave** seria criar uma variável local para evitar o uso de "(*C)".

```
void TransladaCirculo(Circulo *C, float Dx, float Dy)
{
    Circulo A = *C;
    A.centro.x += Dx;
    A.centro.y += Dy;
}
```

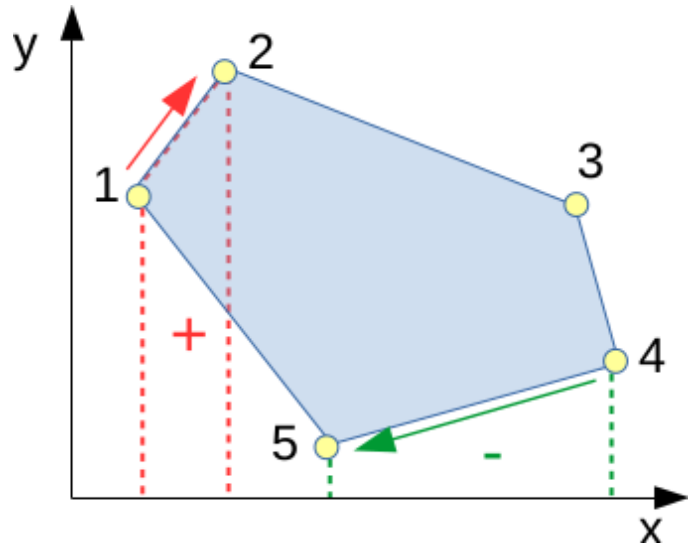
A translação seria perdida com a destruição de "A"ao final da função.

Criando, manipulando, e destruindo polígonos

Um polígono é uma sequência ordenada de vértices (pontos).

```
typedef struct _ponto {
  float x, y; // coordenadas 2D
} Ponto;

typedef struct _poligono {
  Ponto *vertice; // lista de vértices
  int nvertices;  // número de vértices
} Poligono;
```



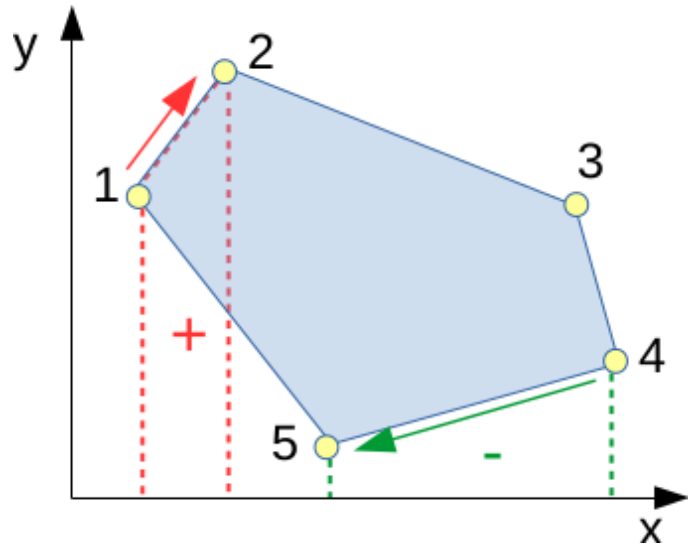
A soma dos comprimentos dos lados define o perímetro e a soma das áreas dos trapézios formados entre cada par de vértices consecutivos define sua área.

Criando, manipulando, e destruindo polígonos

Um polígono é uma sequência ordenada de vértices (pontos).

```
typedef struct _ponto {
  float x, y; // coordenadas 2D
} Ponto;

typedef struct _poligono {
  Ponto *vertice; // lista de vértices
  int nvertices;  // número de vértices
} Poligono;
```



A soma dos comprimentos dos lados define o perímetro e a soma das áreas dos trapézios formados entre cada par de vértices consecutivos define sua área.

Complete o código de poligono.c a partir desta explicação.