


Laboratório de Engenharia de Software

Aula 17

Implementação da programação modular

Francisco Dantas e Alessandro Garcia
LES/DI/PUC-Rio
Outubro 2012

Slides adaptados de: Staa, A.v. Notas de Aula em Programacao Modular; 2008.



Especificação

Laboratório de Engenharia de Software


- Objetivo dessa aula
 - Apresentar os conceitos relacionados com espaços de dados e tipagem de espaços de dados.
- Referência básica:
 - Capítulo 6 do livro texto

Ago 2008

Arndt von Staa © LES/DI/PUC-Rio

2 / 38

Onde são armazenados os dados?




Laboratório de Engenharia de Software

- São armazenados em **espaços de dados**
 - dados
 - código
 - não deixa de ser uma forma de dado

Ago 2008
Arndt von Staa © LES/DI/PUC-Rio
3 / 38

Quais são os atributos de espaços de dados?



Laboratório de Engenharia de Software

- Espaços de dados **contíguos**
 - são armazenados em um **meio de armazenamento**, exemplos:
 - **memória real** (principal)
 - **memória virtual**
 - para todos os processos um único grande arquivo contém todos os dados que estão em memória
 - » quando necessário os dados são transferidos para memória real
 - cada processo tem uma origem nesse arquivo
 - endereço é dado por < idProcesso , inxPagina , Deslocamento >
 - **memória virtual segmentada**
 - arquivos (segmentos) contém os dados que irão para (ou estão em) memória real
 - cada processo identifica n >= 1 segmentos (arquivos)
 - endereço é dado por < idprocesso , idSegmento , inxPagina , Deslocamento >
 - arquivos
 - máquinas remotas

Ago 2008
Arndt von Staa © LES/DI/PUC-Rio
4 / 38

Quais são os atributos de espaços de dados?

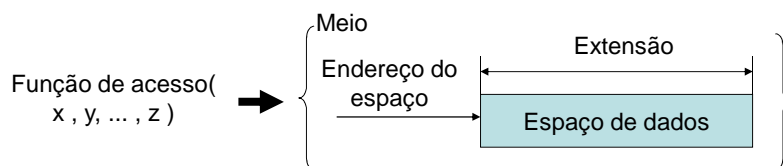


- Espaços de dados **contíguos**
 - possuem uma **extensão**
 - espaço ocupado em bytes
 - `sizeof(Espaco)` é a função que informa a extensão do espaço
 - `sizeof(int)` é igual a o que?
 - independe da máquina?
 - dado: `long vt[100] ;`
 - `sizeof(vt)` é igual a o que?
 - dado: `typedef struct { int a ; char b ; } tpX ;`
 - `sizeof(tpX)` é igual a o que?

Quais são os atributos de espaços de dados?



- Espaços de dados **contíguos**
 - possuem um **endereço real** (virtual) da sua origem
 - são acessados por meio de uma **função de acesso**
 - a forma mais simples dessa função é o **endereço físico do espaço** em memória real (virtual)
 - pode ser uma expressão computacional simples, exemplo
 - `vtAlgo[inxAlgo]`
 - pode ser uma expressão complexa, exemplo:
 - `TabSimb[ObterHash(Simbolo)] -> ValorElemento`
 - acessa o espaço ocupado pelo valor do 1o. elemento da lista de colisão de `simbolo` contido em um tabela de randomização (*hash table*) `TabSim`



Linguagens e espaços de dados



- Sempre que se utiliza uma **linguagem de programação simbólica**, espaços podem ser referenciados por um **nome**, exemplos:
 - nome de uma variável
 - nome de uma constante
 - nome de uma função
- Um nome é, portanto, um parâmetro para uma função de acesso

Linguagens e espaços de dados



- **Amarração** (*binding*) é a operação de associar um nome ou uma função de acesso a um espaço de dados
 - pode ser realizado pelo compilador, exemplos:
 - `int x ;`
 - `int f(int x) { return x * x ; }`
 - pode ser realizado por programa, exemplos:
 - `pX = malloc(dimX) ;`
 - `pV = &V ;`

Espaços aninhados



- **Aninhamento homogêneo**, vetores, matrizes
- Sejam:

```
typedef struct { ... } tpA ;
tpA vtA[ dimA ]
```
- Qual é o `sizeof` de:
 - `vtA[inxA]` ?
 - `sizeof(tpA)`
 - `*vtA` ?
 - `sizeof(tpA)` corresponde a `vtA[0]`
 - `vtA + inxA` ?
 - `sizeof(tpA)` corresponde a `vtA[inxA]`
 - `vtA` ?
 - `sizeof(tpA) * dimA`

Ago 2008

Arndt von Staa © LES/DI/PUC-Rio

9 / 38

Espaços aninhados



- **Aninhamento heterogêneo**
 - estruturas (`struct`) podem conter elementos que por sua vez também são estruturas
- Exemplo de `structs` aninhados em C

```
typedef struct
{
    char Nome[      DIM_NOME ] ;
    char SobreNome[ DIM_NOME ] ;
} tpNome ;

typedef struct
{
    tpNome Nome ;
    tpEndereco Endereco ;
} tpPessoa ;
```

Ago 2008

Arndt von Staa © LES/DI/PUC-Rio

10 / 38

Espaços de dados compostos



- Espaços de dados podem ser
 - **simples**: todo o valor está contido em um único espaço contíguo
 - variável
 - struct
 - vetor
 - lista contida em vetor
 - **compostos**: o valor completo requer vários espaços de dados, simples ou compostos, para ser armazenado
 - lista encadeada
 - árvore
 - base de dados

Espaços de dados compostos



- Espaços de dados compostos podem ser **tratados como um todo** através de um **nome**, exemplos
 - nome associado à **cabeça do espaço de dados composto**
 - nome simbólico que referencia um arquivo
 - nome ODBC que referencia uma base de dados como um todo
 - *handle* que identifica uma janela

Espaços de dados compostos



- Considere uma árvore, qual a diferença entre uma árvore com cabeça e uma árvore identificada pela referência a sua raiz?
 - *Sem cabeça*: o projeto da estrutura estará visível para os clientes,
 - *com cabeça*: encapsula a implementação
 - *Sem cabeça*: se for permitido que várias funções compartilhem uma mesma árvore, esta não poderá ficar vazia, tampouco poderá receber uma nova raiz
 - *com cabeça*: todas as referências são para a cabeça
 - *Com cabeça*: dificulta a exploração
 - *sem cabeça*: percorrimento recursivo convencional
 - *Com cabeça*: facilita a implementação de operadores que atuem sobre a árvore como um todo, exemplos:
 - cópia
 - destruição
 - percorrimento usando um iterador de percorrimento


O que é um tipo de dados?



- Do ponto de vista de quase todos os computadores o **conteúdo** de um espaço de dados é **amorfo**
 - **seqüência de bytes**, mais precisamente seqüência de bits
- A **interpretação** do conteúdo do espaço implica o código a ser executado ao acessá-lo, exemplos
 - $1 + 5 \rightarrow$ integer add
 - $1.0 + 5 \rightarrow$ floating point add, após converter 5 para 5.0
- O **tipo de um espaço** de dados determina como **interpretar** o seu conteúdo:
 - a organização de baixo nível deste espaço
 - conjunto de elementos, caso existam (**struct**)
 - o tamanho em bytes do todo e dos elementos

Laboratório de Engenharia de Software

Tipos de dados




- **Tipos computacionais** são definidos pela linguagem de programação
 - int
 - char
 - char *
- **Tipos do usuário** são declarados pelo programador. Em C:
 - enum
 - struct
 - union
 - typedef
 - **typedef** é na realidade uma espécie de **#define** que conhece a estrutura de uma declaração de tipo do usuário

Ago 2008
Arndt von Staa © LES/DI/PUC-Rio
15 / 38

Laboratório de Engenharia de Software

Espaços de dados contíguos e tipos



- O **tipo** de um espaço de dados contíguo determina **como deve ser interpretado** o seu conteúdo
 - o tipo pode ser **declarado explicitamente** através de código de declaração
 - C, C++, Java
 - o tipo pode ser **declarado implicitamente** por contexto
 - Scheme, Lua
 - funções C para as quais não se tenha declarado um protótipo
 - o tipo pode ser **determinado através de um atributo** contido no espaço de dados
 - identificador do tipo
 - Lua
 - tipos dinâmicos em C++ e Java
 - o tipo pode ser estabelecido **através do código** usado
 - assembler

Ago 2008
Arndt von Staa © LES/DI/PUC-Rio
16 / 38

Dados do tipo union



- **union** define **tipos alternativos** para um mesmo espaço de dados
 - possibilita acessar uma mesma área de dados utilizando diferentes tipos de um conjunto explicitamente especificado
- ```
typedef union tagNome
{
 tipo1 NomeInterpretação1 ;
 tipo2 NomeInterpretação2 ;
 . . .
 tipon NomeInterpretaçãon ;
} tpNome ;
```
- ao acessar o espaço de dados usando **NomeInterpretação<sub>i</sub>** será utilizado o tipo **tipo<sub>i</sub>** para interpretar o espaço
    - o tamanho do espaço de dados da **union tagNome** será igual ao maior dos tamanhos dos tipos
    - o uso descuidado de **unions** pode trazer consequências desastrosas

Ago 2008

Arndt von Staa © LES/DI/PUC-Rio

17 / 38

## Imposição de tipos



- Podem-se forçar diferentes interpretações para um mesmo espaço de dados através da **imposição de tipos** (*type casts*)
- uma imposição de tipos é a uma **union** na qual o **conjunto dos possíveis tipos não é definido a priori**, exemplo

```
(short int *) pEspaco
```

  - estabelece que o espaço de dados designado por **pEspaco** deve ser interpretado como um ponteiro para um **short int** independentemente do tipo com que foi declarado

Ago 2008

Arndt von Staa © LES/DI/PUC-Rio

18 / 38

## Imposição de tipos



- A imposição de tipos em C **pode violar as regras de controle de tipos**, por exemplo
  - no código a seguir “vetor de 4 caracteres” recebe a interpretação “vírgula flutuante” o que provavelmente não faz sentido algum

```
char VetorChar[4] = "abcd" ;
*((float *) VetorChar)
```

## Imposição de tipos, exemplo



```
#include <stdio.h>
#include <memory.h>
void main(void)
{
 char Vet[5] ;
 int c0, c1, c2, c3 ;
 /* Definir o valor contido no espaço de dados */
 memset(Vet , 0 , sizeof(Vet)) ;
 memcpy(Vet , "ABCD" , 4) ;
 /* Exibir várias interpretações do espaço de dados */
 printf("\nlong hexa %lx" , *((long *) Vet)) ;
 printf("\nshort hexa %4x %4x" ,
 *((short int *) Vet) ,
 *((short int *)) &(Vet[2])) ;
 c0 = Vet[0] ;
 c1 = Vet[1] ;
 c2 = Vet[2] ;
 c3 = Vet[3] ;
 printf("\nchar hexa %2x %2x %2x %2x" ,
 c0, c1, c2, c3) ;
 printf("\nlong int %li" , *((long *) Vet)) ;
 printf("\nfloat %f" , *((float *) Vet)) ;
 printf("\nstring %s" , Vet) ;
}
```

## Imposição de tipos, exemplo



- Exercício: ver o resultado da execução

## Quando usar imposição de tipos



- Justifica-se o uso de imposição de tipos irrestrita ao implementar funções de gerenciamento de memória

```
tpMeuTipo * ptMeuTipo ;

. . .
ptMeuTipo = (tpMeuTipo *) malloc(sizeof(tpMeuTipo)) ;
```
- Ao programar em C++ use sempre os operadores `new` e `delete` e jamais as funções `malloc` e `free` mesmo para `structs`

## Quando usar imposição de tipos



- Ao desenvolver módulos genéricos muitas vezes é necessário utilizar **referências** para dados **de tipos desconhecidos**
  - freqüentemente isto requer ponteiros para o **tipo indefinido** (`void *`)
    - em Java o tipo indefinido seria `Object`
    - em C++ pode-se criar uma estrutura de herança
- Existem mecanismos que permitem evitar o uso de `void *`
  - No módulo de definição `zzz.h` inclua exatamente:

```
typedef struct tagXXX * tpXXX ;
```
  - No módulo de implementação `zzz.c` inclua a implementação:

```
struct tagXXX
{
 . . .
} ;
```

Ago 2008

Arndt von Staa © LES/DI/PUC-Rio

23 / 38

## Imposição versus conversão de tipos



- Imposição de tipos e **conversão** de tipos são operações distintas.
  - a imposição de tipos **muda simplesmente a interpretação** do espaço de dados, sem alterar o seu conteúdo
  - consiste meramente em uma instrução para o compilador
- Na conversão, opera-se sobre o conteúdo do espaço de dados
  - transforma o valor no tipo origem para um valor no tipo destino
  - **preserva a semântica do valor**

Ago 2008

Arndt von Staa © LES/DI/PUC-Rio

24 / 38

## Imposição versus conversão de tipos



- Por exemplo ao converter um valor **inteiro** para um valor **vírgula flutuante**
  - usando aritmética de inteiros e manipulação de bits, computa-se uma representação flutuante com o mesmo valor do inteiro
  - ao terminar, atribui-se o valor resultante a uma variável flutuante usando imposição de `float` ao resultado
- Pode-se converter uma **árvore binária** para um **string**
  - percorrer a árvore em ordem pre-fixada pela esquerda
  - a cada nó visitado concatenar o valor do nó seguido de um separador
  - a cada referência `NULL` para sub-árvore concatenar `NIL` seguido do mesmo separador
- O **string** assim formado pode ser convertido de volta para a mesma árvore – façam o experimento ☺

## Cuidados com a imposição de tipos



- **Evite o uso de imposições irrestritas de tipos**
  - Quando utilizar
    - justifique muito bem
      - necessidade
      - correteude
    - inclua comentários no programa contendo esta justificativa
- Evite de todas as maneiras o uso de imposição de tipos **indiscriminada** ao programar **orientado a objetos**
  - vale impor para superclasse, é automático
  - vale impor para classe herdeira somente se tiver certeza que objeto havia sido criado na subclasse imposta ou em uma classe herdeira dela
    - use sempre `dynamic_cast` em C++

Laboratório de Engenharia de Software

LES

FIM

Ago 2008Arndt von Staa © LES/DI/PUC-Rio27 / 38