Estruturação da Informação

Computadores servem para armazenar informação e programas para manipulá-los (não podemos ignorar a importância dos dados e como estruturá-los).

Em um projeto de software, os dois principais aspectos que devem ser estudados são: quais os procedimentos e sobre que dados esses procedimentos atuam.

Desenvolvimento de Programas

O processo de solução de um problema por meio do computador pode ser constituído das seguintes etapas:

(a) especificação do problema;
(b) projeto em alto nível;
(c) análise de alternativas;
(d) refinamento e codificação;
(e) verificação do comportamento.

(a) – é o entendimento das relações existentes entre os dados que são relevantes para o problema, o que sugere uma estruturação lógica para os dados.

(b) – decidir que transformações serão efetuadas sobre os dados no algoritmo para resolver o problema.

(c) – sugerir variações no projeto inicial.

(d) – refinar as transformações do projeto inicial em termos de mecanismos disponíveis na linguagem em que o programa será codificado.

(e) – avaliar o programa obtido para vermos se satisfaz as especificações do problema e quanto ao desempenho (tempo e memória), modificando-o se for o caso.

Três pontos importantes devem ser ressaltados:

- Estruturação de dados;
- Operações;
- Estrutura de representação.

Uma estrutura de dados retrata as relações lógicas existentes entre os dados. Vamos manipular estas estruturas de dados através de operações que as transformam. Tanto as estruturas como as operações precisam ser representados no computador. Estas representações devem:

- Preservar as relações lógicas existentes entre os dados;
- Permitir que as operações sejam descritas por procedimentos simples e eficientes.

Resumindo, um programa é encarado como um algoritmo que manipula dados.

Tipos de Dados

De uma forma geral diferenciamos os tipos de dados pelo conjunto de valores que podem assumir e pelo conjunto de operações que podemos efetuar com eles.

Primitivos, Estáticos e Dinâmicos

Primitivos

São aqueles a partir dos quais podemos definir os demais tipos, não importando como são implementados e manipulados.

Inteiro: representa uma quantidade contável de objetos.
Operações: soma (+), subtração (-), multiplicação (*), divisão (div), resto (mod).
Além disso, podemos comparar inteiros se são iguais (=), diferentes (<>) ou segundo a ordem (<, <=, >, >=).

Real: representa um valor que pode ser fracionado (parte inteira e parte fracionária).
Operações: soma(+), subtração (-), multiplicação (*), divisão (/).
Além disso, podemos comparar dois elementos do tipo real conforme =, <>, <, <=, >, >=.

Lógico: representa dois estados.
Operações: conjunção (E, &), disjunção (OU, |), negação (NÃO, !).
Podemos comparar elementos lógicos quanto a igualdade ou a desigualdade.

Caracter: representa uma seqüência de dígitos, letras e sinais.
Operações: igualdade, diferença, concatenação.

Ponteiro: representa o endereço de um dado na memória.
Operações: igualdade, diferença, soma, subtração, multiplicação, divisão.

Estáticos

São aqueles que têm a estrutura completamente definida antes de começarmos a efetuar as operações. Tipo estático de dado não poderá conter mais elementos do que o previsto inicialmente. Exemplo: vetor.

Dinâmicos

São aqueles que sofrem alteração estrutural quando estão sendo manipulados, à medida que ocorrem inserções e retiradas de elementos. Esse tipo de dado não tem tamanho predefinido (ficando limitado à memória do computador).

Operações

Qualquer que seja o tipo de dado, existem operações clássicas que podemos efetuar.

Criação: primeira operação que devemos efetuar com qualquer tipo de dado.

Busca: consiste na operação de selecionar um determinado elemento. Essa operação é realizada com um dos seguintes objetivos: verificar, utilizar ou atualizar o conteúdo de um elemento ou ainda retirar e inserir novos elementos.

Alteração: em alguns casos a operação de busca é efetuada para que em seguida seja feita uma alteração no conteúdo do elemento encontrado.

Retirada: no caso de estruturas estáticas não podemos retirar um elemento, no máximo podemos retirar os valores que ele contém, substituindo-os por outros. Já em uma estrutura dinâmica podemos retirar quantos elementos desejarmos, onde a cada retirada diminuímos a sua quantidade de elementos.

Inserção: da mesma forma, em uma estrutura dinâmica podemos efetuar a inserção de novos elementos, o que não é possível em uma estrutura estática. A inserção aumenta a quantidade de elementos de uma estrutura.

Funções de Transferência

Em muitos casos basta considerar operações que manipulam valores de um determinado tipo e produzem resultados no mesmo tipo (Ex: soma, multiplicação, negação). Porém é bastante conveniente admitir operações em que mais de um tipo é envolvido. Comparações devem ser encaradas como operações deste gênero, dando o resultado em tipo lógico.

Outros exemplos de funções de transferência:

- Converter um número real em inteiro;
- Converter um número inteiro em real;
- Converter um caracter em inteiro.

Mecanismo para Construção de Tipos

É a construção de um outro tipo de dado a partir dos tipos primitivos.

Vetor

Permite a construção de um tipo cujos valores são agregados homogêneos de um tamanho definido, ou seja, suas componentes são todas de um mesmo tipo.

O formato de definição através do mecanismo vetor é:

vet[*limite_inferior*..*limite_superior*] de *tipo*

Onde *limite_inferior* e *limite_superior* são constantes do tipo inteiro e *tipo* é o tipo das componentes.

Consideremos o tipo definido da seguinte forma:

tipo horas :: *vet*[1..7] de *int*

Que representa as horas de trabalho, por exemplo [0,7,8,5,9,6,4]. Se fulano é uma variável do tipo horas, então fulano[3] tem valor 8.

Em geral, se *c* designa um objeto do tipo *vet*[*m*..*n*] de *t* e *i* designa um inteiro entre *m* e *n* então *c*[*i*] designa o *i*-ésimo componente de *c* (que é um objeto do tipo *t*). Assim, pode-se selecionar qualquer componente.

Registro

São agregados heterogêneos, ou seja, cujas componentes não são necessariamente de um mesmo tipo. Cujo formato geral é:

$reg(sel_1: tipo_1, sel_2: tipo_2, ..., sel_n: tipo_n)$

Onde cada sel_i é um nome de seletor que vai nos permitir designar o componente correspondente do agregado. Por exemplo:

tipo funcionário :: *reg*(cargo: *car*, salário: *int*)

Se fulano é uma variável do tipo funcionário, *fulano.cargo* representa o cargo e *fulano.salário* representa o salário de fulano.

Sequência

Permite construir um tipo cujos objetos são coleções ordenadas de objetos do tipo dado, sem qualquer limitação no seu tamanho. O formato geral é:

seq de *tipo* (ou *seq tipo*)

Onde *tipo* é o tipo dos componentes da seqüência. Por exemplo:

tipo cadeia :: *seq car*

Suponhamos as seqüências <'r', 'a', 'm'>, <'m'>, <'e', 'm', 'i', 'g', 'r', 'a'>, de comprimentos 3, 1 e 6 respectivamente. Se *x* e *y* são variáveis do tipo cadeia, cujos valores são a 1ª e a 3ª constantes acima, temos:

- *princ x* = 'r'
- *cont x* = <'a', 'm'>
- *y conc x* = <'e', 'm', 'i', 'g', 'r', 'a', 'r', 'a', 'm'>

Alternativa

Uma variável só pode ser de um tipo, no sentido que seus possíveis valores só podem ser objetos deste tipo. No entanto, é conveniente permitir que uma mesma variável possa, em momentos diferentes, ter valores de tipos distintos. Isto será possível por meio do mecanismo de alternativa, que forma a união dos tipos correspondentes. O formato geral é:

$alt(tipo_1 \mid tipo_2 \mid ... \mid tipo_n)$

Por exemplo:

tipo questão :: *alt*(*int* | *car*)

Se resposta é uma variável do tipo questão, seu valor pode ser um inteiro ou um caracter.

Referência

Permitirá uma modalidade dinâmica de alocação. Obedece ao formato geral:

ref tipo

Por exemplo:

tipo matéria :: *ref int*

Suponhamos que matemática seja uma variável do tipo matéria. O espaço necessário para a variável matemática compreende duas partes:

- 1ª parte do valor: para armazenar um valor do tipo int;
- 2ª parte de posição: para armazenar uma indicação da localização da parte de valor.

A simples declaração da variável matemática como sendo do tipo matéria ocasiona a alocação de espaço para a parte de posição, mas não ainda para a parte de valor. Para isso precisamos do comando:

aloque matemática

Que faz a alocação da parte de valor e faz com que na parte de posição seja colocada uma indicação da localização onde está alocada a parte de valor.

O espaço alocado pela comando aloque pode ser liberado pela execução de:

desaloque matemática

Enumeração

Permite definir tipos de dados por meio dos valores que os dados daquele tipo podem tomar. A definição é feita indicando-se um conjunto ordenado de identificadores que denotam os valores que os dados daquele tipo devem tomar. O formato geral é:

(id_1, id_2,..., id_n)

Por exemplo, a definição do tipo mês, que é feita por enumeração dos valores válidos que uma variável deste tipo pode ter é:

tipo mês :: (jan, fev, mar, abr, mai, jun, jul, ago, set, out, nov, dez)

Tipo Abstrato de Dados

Um TAD é descrito pela finalidade do tipo e de suas operações, e não pela forma como está implementado.

Um TAD consiste de um novo tipo de dados juntamente com operações que manipulam esses dados. Se a implementação for modificada os programas que utilizam o TAD não precisam ser alteradas.

Criando um tipo abstrato, podemos “esconder” a estratégia de implementação. Quem usa o tipo abstrato precisa apenas conhecer a funcionalidade que ele implementa, não a forma como ele é implementado. Isso facilita a manutenção e o reuso de códigos.

Por exemplo, para trabalharmos com uma lista de números inteiros, precisamos das seguintes operações:

- Cria lista vazia.
- Insere número na lista.
- Retira número da lista.
- Busca número na lista.
- Destrói lista.
-

Isso cria uma interface para a utilização de listas, porém não fala de como será essa implementação.

Outro exemplo seria a estrutura de dados pilha. O suporte necessário seria como empilhar dados na pilha e como desempilhar dados da pilha. Não importando se a estrutura foi feito com alocação estática ou dinâmica.