Estrutura de Dados

Hamilton José Brumatto

Bacharelado em Ciências da Computação - UESC

7 de abril de 2016

Introdução a Estrutura de Dados

Memórias: Bits e Bytes. Como os dados são representados no computador:

- O armazenamento é em um espaço limitado de memória, notação binária.
- O tamanho da estrutura representa o número de elementos representados.
- Um espaço de memória de 1 Byte (8 bits) permite $2^8(256)$ valores representados.
  - Inteiro sem sinal: 0 a 255
  - Inteiro com sinal (primeiro bit é sinal): -128 a 127
  - Símbolos: 'A' ... 'Z', 'a' ... 'z', '0' ... '9', ...
  - Real (ponto flutuante): 1 bit para sinal da mantissa, 1 bit para sinal do expoente, alguns bits para mantissa, alguns bits para expoente. 8 bits fica pouco, melhor 16!
- O tamanho básico da memória depende da arquitetura do processador: 8 bits, 16 bits, ... 64 bits.

Inteiros:

Sem sinal: Notação na base de 2: $00110101 = 0.2^7 + 0.2^6 + 1.2^5 + 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0 = 53$

Com sinal: Complemento de 2:

- Se o primeiro bit é 0, a notação é igual ao exemplo "sem sinal".
- Se o primeiro bit é 1, para identificar o valor, inverte-se os bits, soma-se 1 e aplica a regra acima:
  $10011010 = (\text{Complemento de 2})\ (-)(01100101 + 1) = (-)(01100110) = (-)(2^6 + 2^5 + 2^2 + 2^1) = -102$

Reais: (ponto flutuante)

Vamos considerar 32 bits, não se fala em números reais com menos bits:
24 bits para a mantissa em uma representação de complemento de 2.
8 bits para o expoente em uma representação de complemento de 2.
Escolha da mantissa: 100 pode ser representado como 1×10^2 ou 10×10^1. Vamos escolher de forma que a mantissa não tenha (na base 10) zeros à direita, ou seja, 1×10^2.
Alguns exemplos:

0:	00000000000000000000000000000000
100:	00000000000000000000000100000010
0.03:	00000000000000000000001111111110
123.400:	00000000000001001101001000000010

Quais os limites?

Caracteres e Símbolos:

Padrão ASCII: 8 bits (1 Byte)

Definição de Byte é o número de bits usados para representar um caracter

0	NULL	16	DLE	32	␣	48	0	64	@	80	P	96	‘	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	”	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	’	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	TAB	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	−	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

UTF8-Unicode é uma representação de 2 Bytes para caracteres, permite a maioria dos caracteres acentuados e símbolos em outras línguas como cirílico, grego. Também caracteres especiais.

UTF8-Unicode não é uma representação de tipo primitivo

Operações em Tipos Primitivos

O processador (e linguagens de programação) já oferecem operações básicas sobre tipos primitivos.

```
int i, j, k;
double a, b, c;
char s, t;
unsigned int u, v;
long double x, y, z;
j = 1;
k = j + 2;
s = 'Z';
t = s - k;
```

Seriam todas as operações acima válida? **char** - **int** ???
É possível misturar tipos: double, int, unsigned int, char???

Operações em Tipos Primitivos

O processador (e linguagens de programação) já oferecem operações básicas sobre tipos primitivos.

```
int i, j, k;
double a, b, c;
char s, t;
unsigned int u, v;
long double x, y, z;
j = 1;
k = j + 2;
s = 'Z';
t = s - k;
```

Seriam todas as operações acima válida? **char** - **int** ???
É possível misturar tipos: double, int, unsigned int, char???

Como representar um número complexo: $c = x + y\mathbf{i}$?

Podemos criar uma estrutura que contenha dois valores reais representando a parte real e imaginária do número

```
struct complexo {
    double x, y;
} c;
```

Na manipulação precisamos indicar cada elemento da estrutura:

```
c.x = 3;
c.y = 5;                    // representa c = 3 + 5i
```

A estrutura em si não é um tipo, mas usando o `typedef` podemos definir um tipo com base na estrutura.
Em outras linguagens este recurso pode aparecer com outras estruturas, como o `record` no pascal/delphi.

Classes

Em linguagem orientadas a objetos, o tipo Classe representa um dado semelhante à estrutura, com o detalhe de que além dos dados internos da estrutura, também associamos funções aos elementos da estrutura.

```
class complexo {
    \\Elementos
    double x, y;
    \\Funções
    complexo modulo(complexo c) {...}
    \\Operações
    complexo operator=(complexo c) {...}
    complexo operator+(complexo c) {...}
};
```

A implementação pode variar de linguagem para linguagem.

```
complexo a, b, c;
a.x = 3;
a.y = 5;
b = a;
c = a + b;
```

Nem todas linguagens criam a sobreposições de operadores, ficando estes destinados apenas a tipos primitivos.

Unions

- Unions representam um tipo especial de estrutura que agrupa em um mesmo espaço de memória dois tipos diferentes de estruturas.
- Cabe ao programador interpretar como serão acessados os dados nesta estrutura.
- pode ser importante caso uma única representação interna pode ser interpretada de duas ou mais formas distintas.

Ponteiros representam uma atribuição indireta

Os ponteiros são um tipo especial de dado que armazena endereço de memória de um determinado tipo.
É possível associar ao ponteiro um tipo específico de dado ao qual ele está apontando.
Como é uma representação indireta, é possível criar vários níveis de indireção.

Exemplo de uso de ponteiros

```
typedef struct complexo {
double x,y;
} complexo_t;
complexo_t c;
complexo_t *pc;
c.x = 5; c.y = 3;
pc -> x = 4; pc -> y = 2;
(*pc).x = 7; (*pc).y = 9;
```

Veja texto distribuído que apresenta uma discussão sobre ponteiros.

Vetor é uma estrutura de dados básica:

- Representa uma sequencia de elementos de forma linear, mapeada diretamente na memória, estática.
- Possui operações e manipulações implementadas em hardware e software, inclusive otimizações.
- Todos elementos são do mesmo tipo.
- Se os elementos do vetor forem do tipo primitivo, o conteúdo do vetor é o valor do elemento.
- Se os elementos forem de um tipo não primitivo, o conteúdo pode ser um endereço da memória onde o elemento se encontra.
- Os elementos são indexados por números inteiros, normalmente a partir do 0.

Exemplo de vetor

```
int x[5];
int *px;
int i;
x[0] = 1; x[1] = 2; x[2] = 3; x[3] = 4; x[4] = 5;
px = x;
px+ = 2;
*px = x[4];
```

Como fica o vetor no final desta operação?

Vetor de caracteres ou *strings*

- Se o tipo de dado no vetor é um caracter, teremos uma sequência de letras que podem representar uma frase ou uma sentença.
- Como o espaço do vetor é maior que algumas palavras ou sentença, teremos caracteres que não são significativos no final do vetor, para não referenciá-los costuma-se colocar um terminador.
- Na linguagem C e outras, o terminador é o caracter '\0' ou simplesmente (int) 0. No pascal a estrutura é mais complexa.
- É oferecido em software uma implementação de bibliotecas especiais para este *strings*

Matrizes e além

Como o vetor pode ser uma coleção de tipos abstratos, os elementos de um vetor pode ser, também, um vetor.
Neste caso falamos de Matrizes. Para acessar um elemento precisamos indexar em qual vetor ele se encontra, no vetor de vetores, e dentro deste vetor qual sua posição, de fato.
Este conceito pode se estender a mais dimensões.

```
int m[3][3];
double x[2][2][2][2][2][2];
m[0][2] = 3;
x[0][0][1][1][0][1] = 2;
```

Construa uma implementação para números complexos, e funções que realizam:

- Soma: **void** soma(complexo num1, complexo num2, complexo soma);
- Produto Escalar: **double** escalar(complexo num1, complexo num2);
- Módulo: **double** modulo(complexo num);

Construa uma implementação para números racionais, e funções que realizam;

- Comparação: **int** compara(racional num1, racional num2);// retorna 1 se $num1 < num2$, -1 se $num1 > num2$ ou 0 se $num1 = num2$.
- Construa um vetor de números racionais.
- Faça uma rotina que busque o maior e o menor número no vetor (ambos).

1. Resolver a 1ª Lista de Exercícios.