Tipos Compostos de Dados

UNIÕES E ENUMERAÇÕES

Judson Santos Santiago

Introdução

- As variáveis e constantes armazenam informações
  - Elas ocupam espaço na memória
  - Possuem um tipo
- Os tipos básicos armazenam valores:

Inteiros

```
char   ch   = 'W';
short  sol  = 25;
int    num  = 45820;
```

Ponto-flutuantes

```
float  taxa = 0.25f;
double peso = 1.729156E5;
```

Introdução

- Porém, com os tipos básicos não é possível armazenar um conjunto de informações
  - Como armazenar o peso de 22 jogadores?

```
float p1  = 80.2;
float p2  = 70.6;
float p3  = 65.5;
...
float p21 = 85.8;
float p22 = 91.0;
```

Criar 22 variáveis diferentes não é a melhor solução.

  - A solução é usar vetores:

```
float peso[22];
```

Introdução

- Com vetores não é possível armazenar um conjunto de informações de tipos diferentes
  - Como armazenar um cadastro completo de 22 jogadores? (nome, idade, altura, peso, gols, etc.)

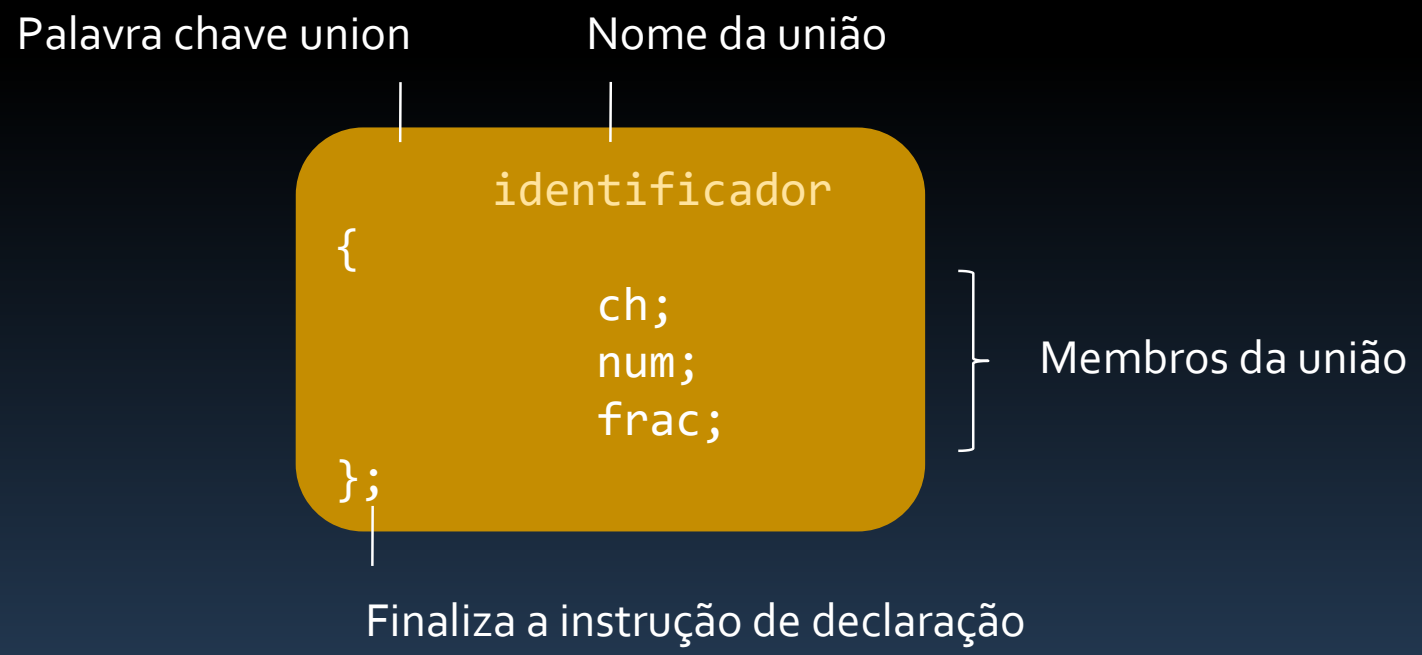
```
char nome[22][80];
unsigned idade[22];
unsigned altura[22];
float peso[22];
unsigned gols[22];
```

Criar vários vetores não é a melhor solução.

  - A solução é usar registros

Uniões

- Assim como um registro, uma união pode armazenar diferentes tipos de dados



Uniões

- A diferença entre um registro e uma união é que a união só armazena um de seus membros por vez
  - O registro armazena um char, um int **e** um double
  - A união armazena um char **ou** um int **ou** um double

```
struct identificador
{
    char   ch;
    int    num;
    double frac;
};
```

```
union identificador
{
    char   ch;
    int    num;
    double frac;
};
```

Uniões

- Os membros compartilham a mesma posição de memória
  - O tamanho do bloco é igual ao do maior membro

```
identificador id;

id.ch = 'a';       // char
cout << id.ch;     // a
id.frac = 3.8;     // double
cout << id.frac;   // 3.8
cout << id.ch;     // lixo
```

```
union identificador
{
    char   ch;
    int    num;
    double frac;
};
```

`id` ocupa 8 bytes,
o mesmo tamanho
de um double

Uniões

```cpp
#include <iostream>
using namespace std;

union CharInt
{
    short num;
    char  ch;
};

int main()
{
    CharInt val = {0};

    cout << "Digite um caractere: ";
    cin >> val.ch;
    cout << "Código ASCII: ";
    cout << val.num << endl;
}
```

// CharInt.cpp

Uniões

- Saída do Programa:

```
Digite um caractere: T
Código ASCII: 84
```

- A inicialização deve fornecer apenas um valor

```
union CharInt
{
    short num;
    char  ch;
};

CharInt val = {0};
```

Uniões

- A união é usada para economizar memória
  - Quando um item pode usar dois ou mais formatos
  - Mas nunca ao mesmo tempo

O número serial de um software pode ser uma chave inteira ou um código de caracteres.

```
union regkey
{
   int   chave;
   char codigo[10];
};
```

```
struct software
{
    char   nome[20];
    float  preco;
    regkey serial;
    bool   tiporg;
};
```

Uniões

```cpp
#include <iostream>
using namespace std;

union regkey  {
     int  chave;
     char codigo[8];
};

int main() {
     cout << "Qual seu tipo de senha?\n[1] chave\n[2] código\nOpção: ";
     int tipo;
     cin >> tipo;

     regkey senha;
     if (tipo == 1) {
        cout << "Digite sua chave: ";  cin >> senha.chave;
     } else {
        cout << "Digite seu código: "; cin >> senha.codigo;
     }
}
```

// RegKey.cpp

Uniões

- Saída do Programa:

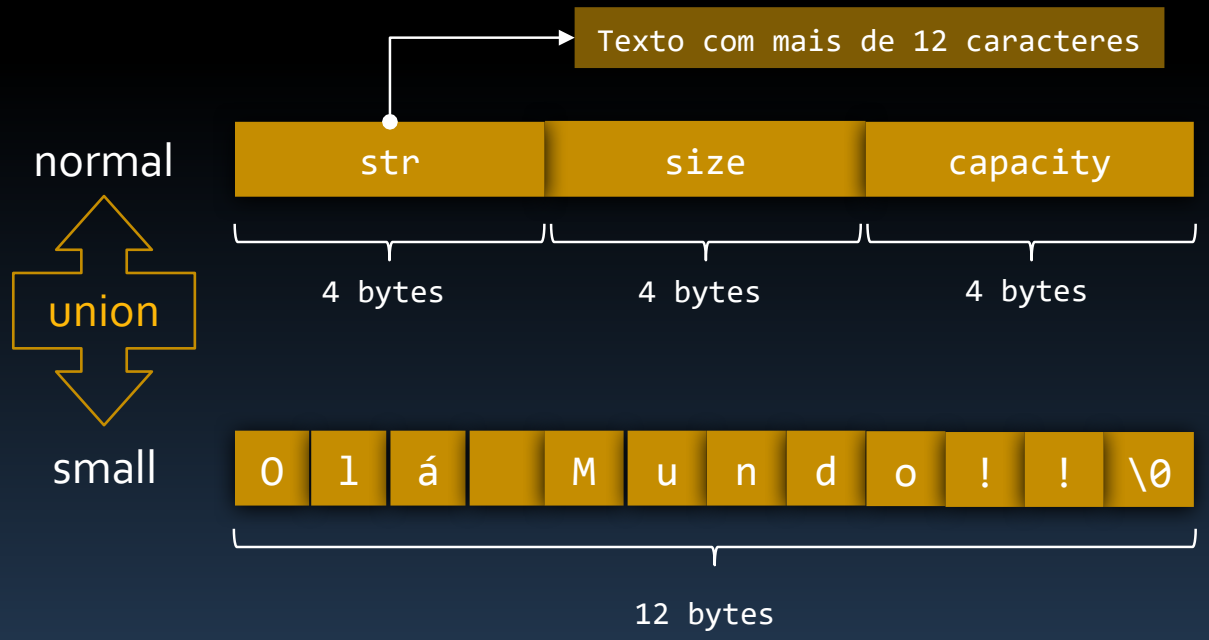
```
Qual seu tipo de senha?
[1] chave
[2] código
Opção: 1
Digite sua chave: 12508
```

- O programador só pode armazenar valores em um dos membros da união, portanto ele deve saber que informação foi digitada

Uniões

- O tipo string internamente pode usar uniões

```
// utiliza otimização para strings pequenas
string nome;
```




```
struct string
{
    union
    {
        struct
        {
            char * str;
            int size;
            int capacity;
        }
        normal;
        char small[12];
    }
    data;
    bool type;
};
```

Enumerações

- Uma enumeração consiste em um conjunto de constantes inteiras, em que cada uma é representada por um nome

```
enum cores {verde, amarelo, azul, branco, preto};
```

- A instrução acima faz duas coisas:
  - Define cores como o nome de um novo tipo
  - Faz dos nomes `verde`, `amarelo`, `azul`, `branco` e `preto` constantes para os valores `0`, `1`, `2`, `3` e `4`

Enumerações

- Fornece uma forma rápida de criar várias constantes

```
enum cores {verde, amarelo, azul, branco, preto};
```

  - A enumeração acima equivale as seguintes declarações:

```
const int verde = 0;
const int amarelo = 1;
const int azul = 2;
const int branco = 3;
const int preto = 4;
```

Enumerações

- Ela é usada quando conhecemos o conjunto de valores que uma variável pode assumir e desejamos usar nomes para esses valores dentro do programa

```
// vermelho = 0, amarelo=1, verde=2, azul=3, preto=4
enum cores {vermelho, amarelo, verde, azul, preto};

// masculino = 0, feminino = 1
enum sexo {masculino, feminino};

// norte = 0, sul = 1, leste = 2, oeste = 3
enum direcao {norte, sul, leste, oeste};
```

Enumerações

```cpp
#include <iostream>
#include <random>
using namespace std;

enum Sexo { Masculino, Feminino };

int main()
{
    cout << "Sorteando o sexo do bebê...\n";

    random_device rand;
    int sorteio = rand() % 2;

    if (sorteio == Masculino)
        cout << "Parabéns, um menino!\n";
    if (sorteio == Feminino)
        cout << "Parabéns, uma menina!\n";
}
```

// Baby.cpp

Enumerações

- Saída do Programa:

```
Sorteando o sexo do bebê...
Parabéns, um menino!
```

- O uso das constantes deixou o código mais claro que:

```
if (sorteio == 0)
    cout << "Parabéns, um menino!\n";
if (sorteio == 1)
    cout << "Parabéns, uma menina!\n";
```

Enumerações

- Se a intenção é criar apenas constantes sem ter um tipo:

```
enum {vermelho, amarelo, verde, azul, preto};
```

- Valores podem ser explicitamente definidos:

```
enum bits {um=1, dois=2, quatro=4, oito=8};
```

- Alguns valores podem ser omitidos:

```
enum bigstep {primeiro, segundo=100, terceiro};
```

- Valores podem ser repetidos:

```
enum {zero, nulo=0, one, um=1};
```

Enumerações

- Após a definição da enumeração, é possível criar variáveis:

```
enum cores {vermelho, amarelo, verde, azul, preto};
cores tinta;
```

  - As únicas atribuições válidas são as de um dos valores definidos na enumeração:

```
   tinta = azul;        // válido
x  tinta = 2000;        // inválido
x  tinta = 3;           // inválido

   tinta = cores (3);   // válido, type cast estilo C++
   tinta = (cores) 3;   // válido, type cast estilo C
   int a = azul;        // válido, conversão automática
```

Enumerações

```cpp
#include <iostream>
using namespace std;

enum mes {Jan=1, Fev, Mar, Abr, Mai, Jun, Jul, Ago, Set, Out, Nov, Dez};

int main()
{
     mes inicio, fim; // cria variáveis do tipo mês

     inicio = Fev;    // inicio do ano letivo
     fim    = Nov;    // fim do ano letivo

     cout << "Digite o número do mês atual: ";
     int atual;
     cin >> atual;    // lê o mês atual para uma variável inteira

     if (atual >= inicio && atual <= fim)
          cout << "Você está em período de aulas.\n";
     else
          cout << "Férias!\n";
}
```

// Meses.cpp

Enumerações

- Saída do Programa:

```
Digite o mês atual: 3
Você está em período de aulas.
```

- As funções de entrada e saída (cin e cout) não sabem como ler ou mostrar um tipo definido pelo programador:

```
    cout << "Digite o mês atual: ";
    mes atual;
x   cin >> atual; // cin não conhece o tipo mes
```

A não ser que sejam ensinadas a fazer isso

Enumerações com Escopo

- As enumerações tradicionais tem alguns problemas:
  - Duas definições podem ter nomes conflitantes:

```
enum pacote { pequeno, grande, largo, jumbo};
enum camisa { pequena, media, grande, extragrande };
```

  - O tipo de um enumerador é dependente da implementação:
    - Eles podem ser constantes de qualquer tipo inteiro
    - Contudo, a partir do C++11 é permitido especificar o tipo

```
enum direcao : short {norte, sul, leste, oeste};
```

Enumerações com Escopo

- As enumerações tradicionais tem alguns problemas:

```
enum cores {vermelho, amarelo, verde, azul, preto};
```

  - Enumeradores são implicitamente convertidos para inteiros:
    - Em atribuições

```
// converte vermelho para 0
int num = vermelho;
```

    - Em comparações

```
// converte preto para 4
if (num < preto)
```

Enumerações com Escopo

- O C++11 resolveu estes problemas com uma nova forma de enumeração que fornece escopo aos enumeradores
  - Enumeradores são de tipo int (quando o tipo não é indicado)

```
enum class pacote { pequeno, grande, largo, jumbo};
enum class camisa { pequena, media, grande, extragrande };

pacote leite = pacote::grande;
camisa promo = camisa::grande;

x int tamanho = camisa::media;      // conversão implícita não permitida
  int carga = int (pacote::jumbo);  // ok, conversão explícita
```

Resumo

- Uniões são semelhantes a registros, mas só armazenam um membro por vez
  - Elas são usadas para economizar memória
  - Especialmente útil em grandes quantidades (vetores)
- Enumerações são usadas para definir constantes inteiras
  - É mais fácil trabalhar com nomes do que com números
  - São usadas quando o número de valores que uma variável pode assumir é conhecido e pequeno