Classes em C++

Declaração de União em C++

Introdução

Uniões em C++ são usadas para armazenar diferentes tipos de dados no mesmo espaço de memória. Diferente de structs, onde cada membro tem seu próprio espaço de memória, em uma união todos os membros compartilham o mesmo espaço, economizando memória.

1. Definição e Sintaxe

- Definição: Uma união é uma estrutura de dados onde todos os membros ocupam o mesmo espaço de memória, permitindo armazenar diferentes tipos de dados, mas apenas um de cada vez.
- Sintaxe:

```
union NomeUniao {
    tipo1 membro1;
    tipo2 membro2;
    // Outros membros
};
```

2. Declaração e Inicialização de Uniões

- Declaração: A declaração de uma união especifica os tipos de membros que podem ser armazenados.
- Inicialização: Uma união pode ser inicializada com um valor para um dos seus membros.

- Exemplo:

```
union Numero {
    int inteiro;
    float flutuante;
};

Numero n;
n.inteiro = 42;
```

3. Acesso aos Membros da União

- Definição: Os membros de uma união são acessados diretamente pelo nome da união seguido do operador de acesso a membro (`.`).
- Exemplo:

```
union Dados {
    int i;
    float f;
    char c;
};

Dados d;
d.i = 10;
std::cout << "Inteiro: " << d.i << std::endl;

d.f = 2.5f;
```

```
std::cout << "Flutuante: " << d.f << std::endl;
```

4. Uniões Anônimas

- Definição: Uniões anônimas são uniões sem nome que permitem acessar diretamente seus membros sem precisar de um objeto da união.
- Exemplo:

```
struct Exemplo {
    union {
        int i;
        float f;
    };
};

Exemplo e;
e.i = 10;
std::cout << "Inteiro: " << e.i << std::endl;
```

5. Uniões e Classes

- Definição: Uniões podem ser usadas dentro de classes para economizar memória quando é necessário armazenar apenas um de vários tipos de dados.
- Exemplo:

```
class Valor {
private:
```

```
    union {
        int i;
        float f;
        char c;
    };
    int tipo;  // 0 para int, 1 para float, 2 para char

public:
    void setInt(int v) { i = v; tipo = 0; }
    void setFloat(float v) { f = v; tipo = 1; }
    void setChar(char v) { c = v; tipo = 2; }

    void print() {
        if (tipo == 0) std::cout << "Int: " << i << std::endl;
        else if (tipo == 2) std::cout << "Float: " << f << std::endl;
        else if (tipo == 2) std::cout << "Char: " << c << std::endl;
    }
};

int main() {
    Valor v;
    v.setInt(10);
    v.print();

    v.setFloat(3.14);
```

```
    v.print();

    v.setChar('a');
    v.print();

    return 0;
}
```

6. Uniões com Membros de Função

- Definição: A partir do C++11, uniões podem ter membros de função, incluindo construtores e destrutores.
- Exemplo:

```
union Valor {
    int i;
    float f;

    Valor() { i = 0; }
    ~Valor() {}
};

int main() {
    Valor v;
    std::cout << "Int: " << v.i << std::endl;
    return 0;
```

```
}
```

7. Limitações das Uniões

- Tamanho do Maior Membro: O tamanho da união é igual ao tamanho do maior membro.
- Inicialização: Apenas um membro pode ser inicializado por vez.
- Acesso a Membros: Acessar um membro diferente do último atribuído resulta em comportamento indefinido.

Dicas de Boas Práticas

- Uso Apropriado: Use uniões quando precisar armazenar diferentes tipos de dados, mas apenas um de cada vez, e quando a economia de memória for crítica.
- Documentação: Documente claramente o uso das uniões para facilitar a manutenção do código.
- Inicialização e Acesso: Inicialize e acesse os membros da união com cuidado para evitar comportamento indefinido.

Esta seção abrange os conceitos sobre declaração de uniões em C++. Para mais detalhes, consulte a documentação oficial: https://en.cppreference.com/w/cpp/language/union