C/C++

Tipos de dados de usuário

Prof. Edson Alves - UnB/FGA

2018

Sumário

Tipos de dados compostos e de usuário

Estruturas

Sintaxe para declaração de estruturas

```
struct identificador {
    declaracao de variaveis, sem valor inicial
};
```

- Uma estrutura em C/C++ consiste numa coleção de variáveis referenciadas por um nome comum
- As variáveis que compõem a estrutura são denominadas membros, campos ou elementos da estrutura.
- É possível definir variáveis do tipo da estrutura criada
- Os membros de uma instância de um estrutura são acessados da seguinte forma: `nome_da_instancia.membro`
- O tamanho de uma estrutura é dado pela soma do tamanho de todos os seus membros, mas pode ser arredondado para cima para o múltiplo mais próximo do tamanho de uma palavra do processador

Exemplo de uso de estrutura

```c
#include <stdio.h>
#include <string.h>

struct Cadastro {
    char nome[128];
    unsigned int idade;
    char sexo;
    float altura;
};

int main() {
    struct Cadastro cadastro;
    strcpy(cadastro.nome, "Maria");
    cadastro.idade = 40;
    cadastro.sexo = 'F';
    cadastro.altura = 1.68f;

    printf("Tamanho da estrutura: %lu\n", sizeof(cadastro));

    return 0;
}
```

Uniões

Sintaxe para declaração de uniões

```
union identificador {
    declaracao de variaveis, sem valor inicial
};
```

- Uma união consiste numa área de memória compartilhada por duas ou mais variáveis
- As variáveis que compõem a união são denominadas membros, campos ou elementos da união
- É possível definir variáveis do tipo da união criada
- Os membros da união são acessados com a mesma sintaxe utilizada nas estruturas
- O tamanho de uma união é igual ao maior dentre todos os tamanhos dos campos da união, e pode ser arredondado de forma semelhante às estruturas

Exemplo de uso de união

```c
#include <stdio.h>

union word32 {
    unsigned int value;
    unsigned char byte[4];
};

int main()
{
    union word32 word, rotate;
    register unsigned int i;

    word.value = 0x12345678;

    for (i = 0; i < 4; i++)
        rotate.byte[i] = word.byte[(i+1) % 4];

    printf("word: %08x, rotate: %08x\n", word.value, rotate.value);

    return 0;
}
```

Classes

Sintaxe para declaração de classes

```
class identificador {
[friend funcao amiga]
[public:]
    [lista de funcoes e variaveis publicas]

[private:]
    [lista de funcoes e variaveis privadas]

[protected:]
    [lista de funcoes e variaveis protegidas]
};
```

Uma classe consiste numa agregação de dados que tem uma relação comum, e num conjunto de funções que agem sobre estes dados.

Classes

- As variáveis que compõem a classe são denominadas membros da classe, enquanto as função são denominadas métodos da classe
- É possível definir variáveis do tipo da classe criada. Estas variáveis são denominadas objetos ou instâncias da classe
- Os membros de uma instância de uma classe ou seus métodos são acessados usando a sintaxe: `instancia.{membro|metodo}`
- Apenas as variáveis e os métodos públicos podem ser acessados fora do escopo da classe
- Funções amigas podem acessar as seções privadas e protegidas da classe

Exemplo de uso de classe

```
1  #ifndef COMPLEX_H
2  #define COMPLEX_H
3
4  #include <ostream>
5
6  class Complex {
7      friend std::ostream& operator<<(std::ostream& os, const Complex& c);
8  public:
9      Complex(double real = 0.0, double imag = 0.0);
10
11     double real() const;
12     double imag() const;
13     void set_real(double real);
14     void set_imag(double imag);
15
16     Complex operator+(const Complex& c);
17 private:
18     double _real, _imag;
19 };
20
21 #endif
```

Exemplo de uso de classe

```cpp
#include <cmath>
#include <complex.h>

using namespace std;

ostream& operator<<(ostream& os, const Complex& c)
{
    os << c.real();

    if (c.imag())
    {
        os << (c.imag() > 0 ? " + " : " - ");
        os << abs(c.imag()) << "i";
    }

    return os;
}

Complex::Complex(double r, double i) : _real(r), _imag(i)
{
}
```

Exemplo de uso de classe

```
23 double Complex::real() const {
24     return _real;
25 }
26
27 double Complex::imag() const {
28     return _imag;
29 }
30
31 void Complex::set_real(double real) {
32     _real = real;
33 }
34
35 void Complex::set_imag(double imag) {
36     _imag = imag;
37 }
38
39 Complex Complex::operator+(const Complex& c)
40 {
41     return Complex(_real + c.real(), _imag + c.imag());
42 }
```

Exemplo de uso de classe

```
#include <iostream>
#include <complex.h>

using namespace std;

int main()
{
    Complex a(1, -1), b(0, 2);

    cout << "a = " << a << endl;
    cout << "b = " << b << endl;

    a.set_real(3);
    b.set_imag(4);

    cout << "a + b = " << a + b << endl;

    return 0;
}
```

Depuração

Depuração

- A depuração (ou *debug*) é o processo de remover ou reduzir o número de erros (*bugs*) em um programa
- Ela abrange desde impressões em pontos-chave do programa ou até programas que acompanham a execução passo-a-passo (depuradores ou *debuggers*)
- Um depurador famoso e livre é o gdb (*GNU Project Debugger*)
- Para que o executável de um código gerado pelo gcc possa ser utilizado pelo gdb, é necessário adicionar a *flag* '-g' na linha de compilação.

Invocando o gdb

- O gdb é um programa em linha de comando, embora existam outros programas que fornecem uma interface gráfica para os comandos do gdb
- O gdb pode ser invocado através do comando

  ```
  $ gdb prog
  ```

  onde prog é o nome do executável a ser depurado
- Caso o executável tenha sido gerado por múltiplos arquivos, a opção '-d' indica o diretório onde se encontram os arquivos-fontes.
- Um executável gerado com a opção de depuração contém informações extra que permitem a depuração, de modo que seu tamanho é, consequentemente, maior do que um executável gerado sem esta opção

Comandos básicos do gdb

`run` inicia a execução do programa

`kill` interrompe a execução, permitindo a reinicialização com um novo *run*

`quit` encerra o gdb

`help` fornece informações sobre o gdb e seus comandos

`list` imprime parte do código-fonte

`break` insere um ponto de parada (*breakpoint*) na linha indicada. Se houverem múltiplos arquivos, o argumento deve ser `nome_do_arquivo:numero_da_linha`. Também pode ser indicado o nome de uma função

Comandos básicos do gdb

`continue`	continua a execução até o próximo *breakpoint*
`next`	avança para a próxima linha do código-fonte
`step`	se houver uma função na linha atual, entra na função
`print`	imprime o valor da variável passada como parâmetro
`watch`	gera informações adicionais quando a variável passada como parâmetro é lida ou escrita
`info break`	lista os *breakpoints*
`info watch`	lista os *watchpoints*

Exemplo para teste de depuração

```
1  #ifndef PRIMES_H
2  #define PRIMES_H
3
4  #ifdef __cplusplus
5  extern "C" {
6  #endif
7
8  extern int is_prime(long n);
9
10 #ifdef __cplusplus
11 }
12 #endif
13
14 #endif
```

Exemplo para teste de depuração

```c
#include "primes.h"
#include <math.h>

int is_prime(long N) {
    long limit, d;

    if (N == 2)
        return 1;

    if (N <= 0|| !(N & 0x01))
        return 0;

    limit = sqrt(N);

    for (d = 3; d <= limit; d++) {
        if (!(N % d))
            return 0;
    }

    return 1;
}
```

Exemplo para teste de depuração

```c
#include <stdio.h>
#include "primes.h"

int main() {
    long xs[] = {2, 3, 5, 7, 11, 13}, ys[] = {-1, 0, 1, 4, 10}, i;

    for (i = 0; i < 6; i++)
        if (!is_prime(xs[i])) {
            printf("Erro: %ld identificado como nao-primo!\n", xs[i]);
            return -1;
        }

    for (i = 0; i < 5; i++)
        if (is_prime(ys[i])) {
            printf("Erro: %ld identificado como primo!\n", ys[i]);
            return -1;
        }

    printf("Test OK!\n");
    return 0;
}
```

Referências

1. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
2. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
3. C++ Reference[1].
4. Página manual do gdb (`$ man gdb`).

[1] https://en.cppreference.com/w/