Escopo de Variáveis
Parâmetros Default
Sobrecarga de Função

Prof. Thiago Felski Pereira, MSc.
Adaptado: Elisangela Maschio de Miranda

Escopo de variáveis

Regras de Escopo de Variáveis

- É o local de um programa em que uma variável é reconhecida. Existem 4 tipos de escopo:
  - Programa.
  - Arquivo-fonte.
  - Função.
  - Bloco.
- Escopo de Programa
  - As variáveis com escopo de programa podem ser referenciadas por qualquer função no programa completo – estas variáveis se chamam variáveis globais. Elas devem ser declaradas fora de qualquer função.

Conselho

Declare as variáveis globais na parte superior do programa. Mesmo podendo ser definidas entre duas funções, declarando-as no princípio do programa fica mais fácil efetuar possíveis mudanças.

Regras de Escopo de Variáveis

- Escopo de arquivo-fonte
  - Variáveis declaradas fora de qualquer função, que possuem a palavra reservada static tem escopo de arquivo-fonte.
  - Estas variáveis podem ser referenciadas do ponto onde estão declaradas até o final do arquivo fonte.

- Escopo de função
  - Só podem ser utilizadas dentro do escopo da função onde estão definidas, ou seja, só são válidas dentro da função onde foram criadas.

- Escopo de bloco
  - As variáveis com escopo de bloco podem ser visualizadas somente dentro do bloco na qual foram declaradas, não sendo visíveis fora deste bloco.

Regras de Escopo de Variáveis

- Variáveis Locais:

  - Possuem um escopo restrito.
  - Elas existem na memória somente quando a função está ativa. Quando a função não está ativa as suas variáveis locais não ocupam espaço na memória, pois não existem.

Detalhes

- No interior de uma função, a menos que mude explicitamente um valor de uma variável, não se pode mudar essa variável por nenhuma sentença externa a função.
- Os nomes das variáveis locais não são únicos. Duas ou mais funções podem definir a mesma variável. Cada variável é distinta e pertence à sua função específica.
- As variáveis locais das funções não existem até que se execute a função. Por isso, múltiplas funções podem compartilhar a mesma memória para suas variáveis locais (mas não ao mesmo tempo).

Regras de Escopo de Variáveis

```
1  #include <iostream>
2  using namespace std;
3
4  int y = 1;
5  int x = 4;
6
7  void primeiro () {
8    static int x = 23;
9    cout<<"\n\n"<<x;
10   x++;
11   cout<<"\n\n"<<x;
12 }
13
14 void segundo () {
15  cout<<"\n\n"<<x;
16  cout<<"\n\n"<<y;
17 }
18
19 void terceiro (int x, int &y) {
20    x *= 2;
21    y *= 3;
22 }
```

```
23
24 int main() {
25    int y = 2;
26    int x = 3;
27
28    cout<<"\n\n"<<y;
29
30    {
31      int x = 10;
32      cout<<"\n\n"<<x;
33      cout<<"\n\n"<<y;
34    }
35
36    primeiro();
37    segundo ();
38    terceiro (x,y);
39    primeiro ();
40
41    cout<<"\n\n"<<x;
42    cout<<"\n\n"<<y;
43
44    return
45 }
```

```
C:\Users\Usuario\Desktop\
Linha 28 do main(): 2
Linha 32 do main(): 10
Linha 33 do main(): 2
Linha 9 da funÞÒo primeiro(): 23
Linha 10 da funÞÒo primeiro(): 24
Linha 15 da funÞÒo segundo(): 4
Linha 16 da funÞÒo segundo(): 1
Linha 9 da funÞÒo primeiro(): 24
Linha 10 da funÞÒo primeiro(): 25
Linha 41 do main(): 3
Linha 42 do main(): 6
```


UNIVALI

Parâmetros Default

Parâmetros Default

Parâmetros Default

- C++ permite a definição de valores padrões para os parâmetros das funções.

- Caso os parâmetros não sejam explicitados estes valores serão utilizados.

```
1   #include <iostream>
2   using namespace std;
3
4   int teste1(int base = 1, int altura = 2) {
5     int area;
6     area = base * altura;
7     return area;
8   }
9
10  int main() {
11    int ba = 10, al = 20;
12    cout<<"\n"<<teste1();
13    cout<<"\n"<<teste1(10);
14    cout<<"\n"<<teste1(ba,al);
15    return 0;
16  }
```


UNIVALI

Parâmetros Default

```
1   #include <iostream>
2   using namespace std;
3   #include <locale.h>
4
5   void sample(char *s = "Sem parâmetros", int i = 0) {
6     cout<<"\nPrimeiro Parametro = "<<s;
7     cout<<"\nSegundo Parametro  = "<<i<<"\n\n";
8   }
9
10  int main() {
11    setlocale(LC_ALL, "Portuguese");
12    sample ();
13    sample("Um parâmetro");
14    sample("Dois parâmetros",10);
15    return 0;
16  }
```

Sobrecarga de função

Sobrecarga de Funções

- C++ possibilita que sejam criadas várias funções com o mesmo nome, desde que tenham conjuntos de parâmetros diferentes (pelo menos quanto ao seu tipo). O compilador identifica a função correta pela análise da quantidade, dos tipos e da ordem dos argumentos na chamada.

- Esse recurso é mais convenientemente utilizado quando as **funções executam operações semelhantes mas possuem lógicas de programação diferentes**.

Sobrecarga de Funções

- **Importante:** ao tentar sobrecarregar uma função modificando somente o tipo retornado será gerado um erro de compilação.

- A versão correta da função será chamada pelo compilador com base nos argumentos recebidos. Por exemplo, ao se criar uma função chamada media() que calcula a média entre dois números int, long, float ou double, a versão correta será chamada conforme os argumentos recebidos, sejam do tipo int, long, float ou double.

Sobrecarga de Funções

```
1  #include <iostream>
2  using namespace std;
3
4  float media (float n1, float n2) {
5    return ((n1+n2)/2.0);
6  }
7
8  float media (float n1, float n2, float n3) {
9    return ((n1+n2+n3)/3.0);
10 }
11
12 int main() {
13   float nota1, nota2, nota3;
14   cout<<"Nota 1 ..: ";
15   cin>>nota1;
16   cout<<"Nota 2 ..: ";
17   cin>>nota2;
18   cout<<"Nota 3 ..: ";
19   cin>>nota3;
20   cout<<"\nMedia com 2 valores ..: "<<media(nota1,nota2)<<"\n";
21   cout<<"\nMedia com 3 valores ..: "<<media(nota1,nota2,nota3)<<"\n";
22   return(0);
23 }
```

Sobrecarga de Funções

```
1   #include <iostream>
2   using namespace std;
3
4   int soma(int a, int b) {
5       return (a+b);
6   }
7
8   int soma(int a, int b, int c) {
9       return (a+b+c);
10  }
11
12  double soma(double a, double b) {
13      return (a+b);
14  }
15
16  int main() {
17      cout << soma(1,2) << endl;
18      cout << soma(3,4,5) << endl;
19      cout << soma(6.7,8.9) << endl;
20      return 0;
21  }
```

Obrigado pela atenção

contato: Felski@univali.br