INTRODUÇÃO AO C++

SISTEMAS DE INFORMAÇÃO

DR. EDNALDO B. PIZZOLATO

Tópicos

- Estrutura Básica de Programas C e C++
- Tipos de Dados
- Variáveis
- Strings
- Entrada e Saída de Dados no C e C++

INTRODUÇÃO

O C++ aceita a sintaxe C, acrescentando melhoramentos, ampliando o escopo de aplicações, e possibilitando o desenvolvimento de programas baseados no paradigma da Orientação a Objetos;

INTRODUÇÃO

O C++ mantém à característica do C de ser uma linguagem de pequeno tamanho!

C++ possui apenas 62 palavras reservadas (32 delas comuns ao C);

INTRODUÇÃO

A diferença maior se dá nas unidades de programação: no C são as funções enquanto que no C++ são as classes (que instanciam objetos e contém funções – chamadas de métodos).

PALAVRAS RESERVADAS EM C E C++

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

PALAVRAS RESERVADAS EM C E C++

new	wchar_t	mutable	explicit	bool
delete	dynamic_cast	asm	public	true
inline	reinterpret_cast	this	protected	false
typeid	static_cast	try	private	
template	const_cast	catch	operator	
using	typename	throw	friend	
namespace	virtual	class		

Modelo

```
// Diretivas do Processador
#include <iostream.h>

void main()
{
      cout << "Meu primeiro programa" << endl;
}
```

Modelo

```
// Diretivas do Processador
#include <iostream.h>

int main()
{
      cout << "Meu primeiro programa" << endl;
      return 0;
}
```

Modelo (compilador ISO98 v1.0)

```
// Diretivas do Processador
#include <iostream.h>

int main()
{
      std::cout << "Meu primeiro programa" << endl;
      return 0;
}
```

Modelo (compilador ISO98 v1.1)

```
// Diretivas do Processador
#include <iostream.h>
using std::cout;
int main()
{
      cout << "Meu primeiro programa" << endl;
      return 0;
}
```

Modelo (compilador ISO98 v1.2)

```
// Diretivas do Processador
#include <iostream.h>
using namespace std;
int main()
{
      cout << "Meu primeiro programa" << endl;
      return 0;
}
```

/* Namespace – faz parte do que se chama ***Ambiente de Nomes*** no C++, que é um mecanismo para expressar agrupamentos lógicos.*/

Algumas palavras sobre comentários

- Documentam programas
- Melhoram a leitura de programas
- São ignorados pelo compilador

Algumas palavras sobre diretivas de pré-processamento

- Começam com #
- Processadas por pré-processador antes da compilação

TIPOS DO C++

TIPO	NOME	modificadores
lógico	bool	
inteiro	int	unsigned, short, long
real	float double	long
literal	char	unsigned
void	-	

TIPOS DO C++

declaração	bits	bytes	faixa
char	8	1	-128 a 127
unsigned char	8	1	0 a 255
int	16/32	2/4	
unsigned int	16/32	2	
short int	16	2	-32768 a 32767
long int	32	4	$-2.14\ 10^9$ a $2.14\ 10^9$
unsigned long	32	4	0 a $4.29\ 10^9$

DECLARAÇÃO DAS VARIÁVEIS

Sintaxe de declaração de variáveis em C e C++ é semelhante.

tipo_de_dado lista_variáveis;

Exemplos :

```
int x, y, z;
float f;
```

REGRAS PARA NOMES DAS VARIÁVEIS

-São reconhecidos pelo C os 32 primeiros caracteres de uma variável;

-Só se pode iniciar um nome com um caractere do alfabeto ou pelo caractere underscore '_';

REGRAS PARA NOMES DAS VARIÁVEIS

-Os demais caracteres podem ser alfabeto, underscore ou caracteres numéricos;

- O C e o C++ consideram diferentes nomes com letras maiúsculas e minúsculas. Portanto as seguintes variáveis são diferentes : int maior, MAIOR, Maior, mAiOr;

LOCAL DAS DECLARAÇÕES DE VARIÁVEIS

Em C, só se pode declarar uma variável dentro de função, antes das instruções ou entre funções (sendo, neste caso, global a partir de sua declaração).

LOCAL DAS DECLARAÇÕES DE VARIÁVEIS

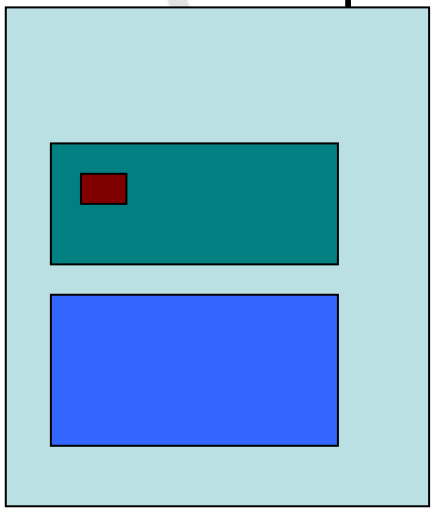
Em C++, além destas formas, pode-se declarar uma variável em qualquer ponto de um programa - entre instruções ou mesmo dentro de instruções.

ESCOPO DE VARIÁVEIS

As regras de escopo referem-se à visibilidade da variável. Há 3 escopos possíveis para uma variável: local, arquivo e classe.

ESCOPO DE VARIÁVEIS

LOCAL: são acessadas única e exclusivamente pelo bloco no qual foram declaradas.

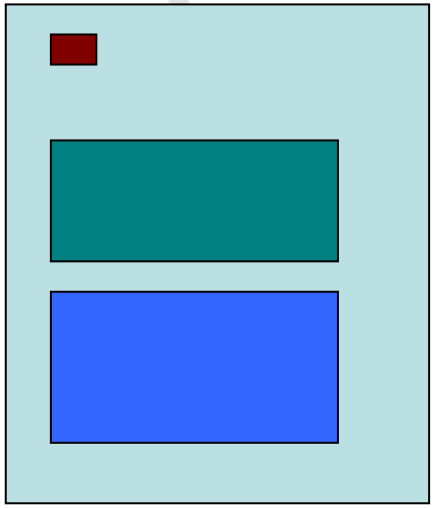


Um bloco é definido por

{....}

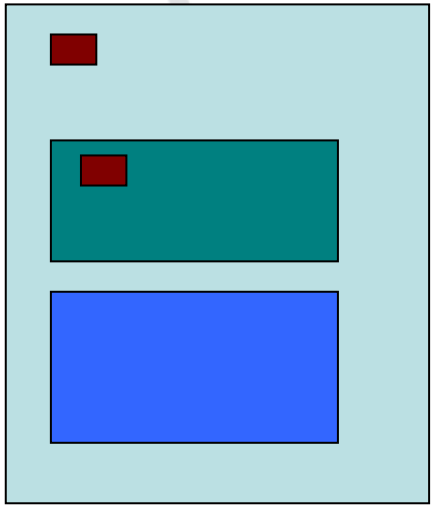
ESCOPO DE VARIÁVEIS

GLOBAL: são acessíveis em todo o arquivo.



ESCOPO DE VARIÁVEIS

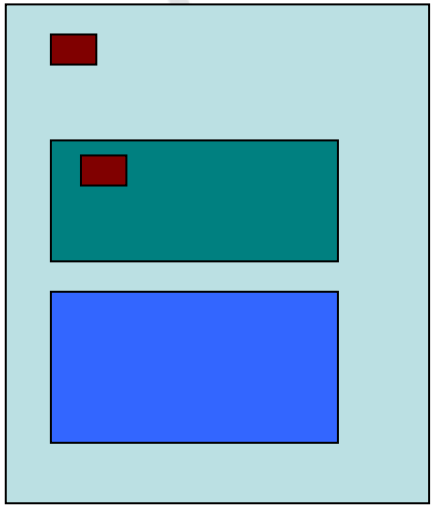
GLOBAL: blocos com variáveis locais homônimas a variáveis globais perdem o acesso às últimas.



ESCOPO DE VARIÁVEIS

O C++ diferencia variáveis locais e globais homônimas com o **Operador de Resolução de Escopo** :: (também chamado Qualificação de Escopo), que colocado em frente à variável, faz com que o programa acesse a global.

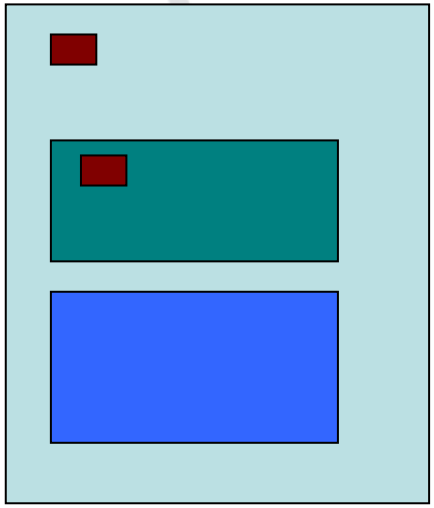
Um erro será apontado, caso não exista a variável global.



ESCOPO DE VARIÁVEIS

```
#include <iostream.h>
 int x = 2;
main ()
{   int x = 0;
    cout << "local " << dob( x );
    cout <<"global"<< dob(::x);
}
int dob (int v)
{
        return 2*v;
}
```

Resultado : local = 0 global = 4



CLASSES DE ARMAZENAMENTO

As classes de armazenamento referem-se ao tempo de vida das variáveis.

C++ não acrescenta novas definições ao C.

CLASSES DE ARMAZENAMENTO

Os especificadores **auto** e **register** significam tempo de vida local, ou seja, são liberados da memória quando o programa encerra seu escopo de declaração.

CLASSES DE ARMAZENAMENTO

Vale lembrar que todas as **variáveis locais** normalmente declaradas em C e C++ são *automáticas*, ou seja, do tipo **auto**.

CLASSES DE ARMAZENAMENTO

Usa-se **register** para otimizar o acesso, solicitando a alocação da variável em algum registrador disponível. Somente variáveis dos tipos **char** e **int** e suas variantes (**signed**, **unsigned**, **long**, etc) podem ser declaradas como **register**.

CLASSES DE ARMAZENAMENTO

Os especificadores **static** e **extern** possuem tempos de vida globais, ou seja, permanecem alocados em memória guardando seu último valor mesmo após o encerramento de seu escopo.

CLASSES DE ARMAZENAMENTO

Usa-se **extern** para se especificar que a definição da variável ocorre uma única vez, em qualquer um dos arquivos fontes que compõem o programa.

CLASSES DE ARMAZENAMENTO

Em C, o especificador **extern** foi idealizado para auxiliar a compilação em separado de arquivos. Em C++, porém, qualquer variável ou função não declarada explicitamente como **static** será, do tipo **extern**.

CLASSES DE ARMAZENAMENTO

A declaração static faz com que a variável permaneça existindo em memória enquanto o programa estiver rodando.

```
int dobro (int x)
{
        static int cont=0;
        cont++;
        printf("%d \n",cont);// não
      fazer isso
  return 2*x;
}
```

CLASSES DE ARMAZENAMENTO

A abrangência da variável não se altera com a declaração **static**.

Por exemplo, **cont** continua sendo uma variável local da função dobro().

CLASSES DE ARMAZENAMENTO

Quando a declaração **static** é usada em variáveis globais, o escopo da variável é modificado única e exclusivamente no arquivo fonte onde ela se encontra.

CONVERSÃO DE TIPOS

Alternativamente à conversão de cast do C, C++ propõe uma conversão explícita de tipo, onde se pode converter um dado tanto para tipos pré-definidos da linguagem como para tipos definidos pelo programador.

CONVERSÃO DE TIPOS

Exemplo :

```
char letra = 'a';
int var = letra; // var vale ASCII de 'a‘
```

CONVERSÃO DE TIPOS

Exemplo :

```
float fReal = 45.60;
int iInteiro, iOutro;
char cCaracter = 'W';

iOutro = (int) cCaracter;
cCaracter = char(fReal);
iInteiro = (int) fReal;
```

CADEIAS DE CARACTERES

Exemplo :

```
char cadeia[20];
string x;  // conteiner – incluir <string.h>

x = “Objetos";
cout << “x : " << x << endl;
```

ENTRADA E SAÍDA DE DADOS

Bibliotecas Stream

entrada de dados pelo teclado

cin (scanf no C)

saída de dados para tela

cout (printf no C)

é necessária a inclusão da biblioteca iostream.h

```
#include <iostream.h>  // no C stdio.h
```

ENTRADA E SAÍDA DE DADOS

Exemplos :

Variáveis	C++	C
int x;	cin >> x;	scanf("%d",&x);
float f,g;	cin >> f >> g;	scanf("%f%f",&f,&g);

ENTRADA E SAÍDA DE DADOS

Exemplos :

Variáveis	*C++*	*C*
int x;	cout << “x = “ << x;	printf("x=%d",x);

ENTRADA E SAÍDA DE DADOS

Atenção :

É possível imprimir mais de uma variável ao mesmo tempo!

cout << x << " e " << y << endl;

MACROS OU DIRETIVAS #DEFINE

Sintaxe :

#define frase1 frase2

- Durante a compilação ao se encontrar a ocorrência de frase1, o compilador substituirá por frase2.

MACROS OU DIRETIVAS #DEFINE

Exemplo :

```
#define PI 3.141596
...
area = PI * raio*raio; (no código)

area = 3.141596 *raio*raio; (código compilado)
```

ESPECIFICADOR CONST

Surgiu como alternativa ao #define na primeira versão do C++ e mais tarde foi incorporado ao padrão ANSI C;

Para se declarar um valor constante, usa-se a palavra const seguida do tipo e do valor da constante;

Em C++, é possível tornar constantes: valores, ponteiros, conteúdo de ponteiros e parâmetros de função.

ESPECIFICADOR CONST

Sintaxe :

const tipo tVar = valor;

- Depois de definido o valor não se pode mais alterá-lo.
- Não ocorre substituições como com define!

Exemplo:

```
const float PI = 3.141596;
```

DEFINIÇÃO DE TIPOS

Pode-se definir tipos próprios de variáveis, combinando tipos já existentes;

Para isto utiliza-se a palavra reservada:

typedef

Exemplo:

typedef int MEU_INT;

Conclusões:

C++ é uma linguagem muito parecida com C.

Quem aprendeu C não terá dificuldades em entender a linguagem C++. Aliás, C, C++, C# e Java são muito parecidas.

Conclusões:

Vimos que C++ incluiu tipos novos (bool, por exemplo) e, agora, permite definir uma constante utilizando const.

Também permite utilizar os valores lógicos true e false.

Conclusões:

Para quem tinha dores de cabeça com os comandos de entrada e saída (scanf e printf) uma boa nova:

cin

E

cout

FIM