Rodrigo Henrich

rodrigohenrich@faccat.br

Escopo e tempo de vida de variáveis

- O escopo e o tempo de vida de uma variável está intimamente ligada a posição dela dentro do código
- Dependendo do local onde elas são declaradas elas podem ser
  - globais
  - locais

Variáveis globais

- Tem por característica existir por toda a execução do programa, ela é acessível em qualquer função do código
- No caso do C, elas são declaradas antes da função main()
- O tempo de vida delas é o mesmo do programa, elas irão existir até que o programa termine

Variáveis globais

```c
#include <stdio.h>

int x = 5;

int main(){
  //Acesso a variável global
  printf("%d",x);
  //Incremento da variável global
  x = x+5;
  return 0;
}
```

Variáveis globais

```c
#include <stdio.h>

int x = 5;

void incremento(){
  x = x+1;
}

int main(){
  //Acesso a variável global
  printf("%d",x);
  //Chamada de função do meu programa
  incremento();
  //Acesso a variável global
  printf("%d",x);
  return 0;
}
```

Escopo local

- Variáveis com escopo local existem apenas dentro do local onde são declaradas e pelo tempo que aquele trecho de código for executado
- Uma variável declarada dentro de uma determinada função só existirá naquela função e pelo tempo que ela for executada.

Escopo local

```c
#include <stdio.h>
int x = 5;
void incremento(){
  //Declarando uma variável local
  int y = 1;
  x = x+y;
}
int main(){
  //Declarando uma variável local
  int a = 10;
  //Acesso a variável local
  printf("\n%d",a);
  //Chamada de função do meu programa
  incremento();
  //Acesso a variável global
  printf("\n%d",x);
  return 0;
}
```

Escopo local

```c
#include <stdio.h>
int main(){
  int a = 10;
  //Acesso a variável local
  printf("\n%d",a);
  //Chamada de função do meu programa
  incremento();
  if(a>=10){
    /**A variável x só vai existir se o IF der verdade
    * e ela será extinda ao final do IF*/
    int x = 2;
    printf("\n%d",x);
  }
  return 0;
}
```

Escopo local

```c
#include <stdio.h>
int g = 0; //Escopo global
int main(){
  int a = 10;//Escopo local
  //Acesso a variável local
  printf("\n%d",a);
  //Chamada de função do meu programa
  incremento();
  if(a>=10){
    int x = 2;//Escopo local dentro de outro escopo local
    printf("\n%d",x);
  }
  return 0;
}
```

Mesmo nome de variável

```c
#include <stdio.h>
int g = 0; //Escopo global
int main(){
  int a = 10;//Escopo local
  //Acesso a variável local
  printf("\n%d",a);
  //Chamada de função do meu programa
  incremento();
  if(a>=10){
    int x = 2;//Escopo local dentro de outro escopo local
    printf("\n%d",x);
  }
  return 0;
}
```

Constantes

- Como sabemos variáveis são espaços de memória onde são armazenados valores que podem ser modificados ao longo do programa
- As constantes são também espaços de memória
- No entanto o valor dela será constante para toda a execução do programa
- Para criar constantes, em c temos duas formas
- Uma constante nada mais é do que usar uma palavra específica para falar de determinado valor
- O comando **#define**
- Ou o comando **const**

#define

- Sua sintaxe

#define nomeConstante valorConstante

```c
#include<stdio.h>
#define PI 3.1415

int main(){
  float raio = 5;
  float area = PI*raio*raio;
  printf("%.2f",area);
  return 0;
}
```

const

- Da mesma forma que o define o const permite criar constantes
- Seu uso é muito semelhante ao de uma declaração de variáveis

const tipo nome_constante = valorConstante

```c
#include<stdio.h>
const float PI = 3.1415;
int main(){
  float raio = 5;
  float area = PI*raio*raio;
  printf("%.2f",area);
  return 0;
}
```

Operadores relacionais

- Assim como usamos nos algoritmos na linguagem de programação C precisamos de operadores relacionais
- Eles nos permitem comparar duas variáveis ou valores
- Esse operador retorna como resultado um valor binário
- 0 - falso, false
- 1 - verdade, true

Operadores relacionais

Operador	Significado	Exemplo
>	maior do que	x > 5
>=	maior ou igual a	x >= 10
<	menor do que	x < 5
<=	menor ou igual a	x <= 10
==	**igual a**	**x == 0**
!=	**diferente de**	**x != 0**

Exemplos

```c
#include<stdio.h>

int main(){
   int x = 5;
   int y = 3;
   printf("%d\n",x > 4);
   printf("%d\n",x == 4);
   printf("%d\n",x != 4);
   printf("%d\n",x != y+2);
}
```

Operadores lógicos

- Se precisarmos estabelecer mais de uma regra em nossas comparações é preciso usar os operadores lógicos,
- Por exemplo 0 < x < 10 o valor de x deve ser maior que 0 e maior que 10
- Para modelar essa situação temos os operadores lógicos em programação.
- E lógico representado em C por &&
- OU lógico representado em C por ||
- Não lógico representado em C por !

Operadores lógicos

Operador	Significado	Exemplo
&&	**E** lógico	(x >= 0 && x <= 9)
||	**OU** lógico	(a == 'F' || b != 32)
!	**Não** lógico	!(x == 10)

Operadores lógicos

- Essas operações também resultam da mesma forma
- 0 - falso, false
- 1 - verdade, true

Operadores lógicos

Tabela verdade					
a	b	!a	!b	a && b	a || b
0	0	1	1	0	0
0	1	1	0	0	1
1	0	0	1	0	1
1	1	0	0	1	1

Operadores de atribuição simplificada

Operador	Significado	Exemplo		
+=	soma a atribui	x += y	equivale a	x = x + y
-=	subtrai e atribui	x -= y		x = x - y
*=	multiplica e atribui	x *= y		x = x * y
/=	divide a atribui	x /= y		x = x / y
%=	divide e atribui o resto	x %= y		x = x % y

Operadores de pré e pós incremento

Operador	pré	pós
++ incremento	++x	x++
-- decremento	--x	x--

Operadores de pré e pós incremento

```c
#include<stdio.h>

int main(){
   int x = 10;
   x++;
   printf("%d",x);
}
```

```c
#include<stdio.h>

int main(){
   int x = 10;
   ++x;
   printf("%d",x);
}
```

Operadores de pré e pós incremento

```c
#include<stdio.h>

int main(){
  int a = 10;
  printf("%d", ++a);
  printf("\n%d", a);
}
```

Imprime
11
11

```c
#include<stdio.h>

int main(){
  int a = 10;
  printf("%d", a++);
  printf("\n%d", a);
}
```

Imprime
10
11

Operadores de pré e pós incremento

```c
#include<stdio.h>

int main(){
   int x = 10;
   x--;
   printf("%d",x);
}
```

```c
#include<stdio.h>

int main(){
   int x = 10;
   --x;
   printf("%d",x);
}
```

Operadores de pré e pós incremento

```c
#include<stdio.h>

int main(){
   int x = 10;
   printf("%d\n",x++);
   printf("%d",x);
}
```

```c
#include<stdio.h>

int main(){
   int x = 10;
   printf("%d\n",++x);
   printf("%d",x);
}
```

Operadores de pré e pós incremento

```c
#include<stdio.h>

int main(){
   int x = 10;
   printf("%d\n",x--);
   printf("%d",x);
}
```

```c
#include<stdio.h>

int main(){
   int x = 10;
   printf("%d\n",--x);
   printf("%d",x);
}
```

Operadores de pré e pós incremento

- Note que o
- x++ equivale a x = x+1
- x-- equivale a x = x-1

Operadores bit a bit

- Os números computacionalmente são representados de forma binária
- Em C é possível realizar comparações e operações com cada um destes bits, obtendo o resultado
- Podemos usar a seguinte tabela para conversão decimal para binário

512	256	128	64	32	16	8	4	2	1

Convertendo números

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0

- Supondo a conversão do número 3 decimal para binário
- Quais bits temos que ligar para gerar esse número?

Convertendo números

128	64	32	16	8	4	2	1
0	0	0	0	0	0	1	1

- Supondo a conversão do número 3 decimal para binário
- 00000011

Convertendo números

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0

- Supondo a conversão do número **32** decimal para binário
- Quais bits temos que ligar para gerar esse número?

Convertendo números

128	64	32	16	8	4	2	1
0	0	1	0	0	0	0	0

- Supondo a conversão do número **32** decimal para binário
- 00100000

Convertendo números

128	64	32	16	8	4	2	1
0	0	0	0	0	0	0	0

- Supondo a conversão do número **45** decimal para binário
- Quais bits temos que ligar para gerar esse número?

Convertendo números

128	64	32	16	8	4	2	1
0	0	1	0	0	0	0	0

- Supondo a conversão do número **45** decimal para binário
- Ligando o bit equivalente ao decimal 32 (45-32=13)
- Ainda temos que converter o decimal 13

Convertendo números

128	64	32	16	8	4	2	1
0	0	1	0	1	0	0	0

- Supondo a conversão do número **45** decimal para binário
- Ligando o bit equivalente ao decimal 8 (13-8 = 5)
- Ainda temos que converter o decimal 5

Convertendo números

128	64	32	16	8	4	2	1
0	0	1	0	1	1	0	0

- Supondo a conversão do número **45** decimal para binário
- Ligando o bit equivalente ao decimal 4 (5-4 = 1)
- Ainda temos que converter o decimal 1

Convertendo números

128	64	32	16	8	4	2	1
0	0	1	0	1	1	0	1

- Supondo a conversão do número **45** decimal para binário
- Ligando o bit equivalente ao decimal 1
- 45 decimal equivale a
- 00101101 em binário

Operadores bit a bit

&	E bib a bit	x & 167
|	OU bit a bit	x | 129
<<	deslocamento de bits a esquerda	x << 2
>>	deslocamento de bits a direita	x >> 2

Complemento bit a bit

- Supondo o número 82

128	64	32	16	8	4	2	1
0	1	0	1	0	0	1	0

- Em binário ele equivale a 01010010
- Aplicando o complemento sobre o número 82
- O resultado seria a inversão dos números
- Onde era 1 fica 0
- Onde era 0 fica 1

E bit a bit

- Supondo o número 82

128	64	32	16	8	4	2	1
0	1	0	1	0	0	1	0

- Em binário ele equivale a 01010010
- E o número 16

128	64	32	16	8	4	2	1
0	0	0	1	0	0	0	0

- Em binário 00010000

E bit a bit

- Aplicando o E bit a bit entre os dois o resultado seria 16

	128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
82	0	1	0	1	0	0	1	0
16	0	0	0	1	0	0	0	0
16	**0**	**0**	**0**	**1**	**0**	**0**	**0**	**0**

E bit a bit

```c
#include<stdio.h>
int main(){
   int a=82;
   int b=a&16;
   printf("%d",b);
   return 0;
}
```

OU bit a bit

- Supondo o número 82

128	64	32	16	8	4	2	1
0	1	0	1	0	0	1	0

- Em binário ele equivale a 01010010
- E o número 40

128	64	32	16	8	4	2	1
0	0	1	0	1	0	0	0

- Em binário 00101001

OU bit a bit

- Aplicando o OU bit a bit entre os dois o resultado seria 40

	128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
82	0	1	0	1	0	0	1	0
40	0	0	1	0	1	0	0	0
122	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**0**

OU bit a bit

```c
#include<stdio.h>
int main(){
   int a=82;
   int b=a|40;
   printf("%d",b);
   return 0;
}
```

Deslocamento de bits a esquerda

- E o número 40

128	64	32	16	8	4	2	1
0	0	1	0	1	0	0	0

- Em binário 00101001

Deslocamento de bits a esquerda

- Aplicando o deslocamento de 2 bits à esquerda

	128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
40	0	0	1	0	1	0	0	0
160	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**

Deslocamento de bits a esquerda

```c
#include<stdio.h>
int main(){
   int a=40;
   int b=a<<2;
   printf("%d",b);
   return 0;
}
```

Deslocamento de bits a direita

- E o número 136

128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

- Em binário 10001000

Deslocamento de bits a direita

- Aplicando o deslocamento de 2 bits à direita

	128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
136	1	0	0	0	1	0	0	0
34	**0**	**0**	**1**	**0**	**0**	**0**	**1**	**0**

Deslocamento de bits a direita

```c
#include<stdio.h>
int main(){
   int a=136;
   int b=a>>2;
   printf("%d",b);
   return 0;
}
```

Modeladores de tipo (cast)

- Como sabemos o resultado da operação entre dois inteiros é um número inteiro;
- Desta forma se tentarmos armazenar o resultado em uma variável float o número será apenas a parte inteira da resposta
- No exemplo ao lado r deveria valer 0.5
- No entanto o resultado é 0 mesmo sendo r uma variável do tipo float

```
1  #include<stdio.h>
2  int main(){
3      int n=1;
4      int d=2;
5      float r;
6      r = n/d;
7      printf("%f",r);
8      return 0;
9  }
```

Modeladores de tipo (cast)

- Para resolver isso é possível realizar casting de variáveis
- Ele tem a seguinte sintaxe
- (**nome_tipo**) expressão
- Modificando o programa para o seguinte o resultado deverá ser mais correto
- Note o exemplo de casting na linha 6

```
1  #include<stdio.h>
2  int main(){
3     int n=1;
4     int d=2;
5     float r;
6     r = (float)n/d;
7     printf("%f",r);
8     return 0;
9  }
```

Estrutura de seleção IF

- O IF permite assim como o SE dos algoritmos tomar determinadas decisões no programa
- Sua sintaxe é simples

```
if(condicao)
    instrucao
```

Estrutura de seleção IF

```
1  #include<stdio.h>
2  int main(){
3     int x = 10;
4
5     if(x<=10)
6        printf("x e menor ou igual a 10");
7  }
```

Estrutura de seleção IF

- Caso tenha uma condição de senão

  if(condicao)

    instrucao

  else

    instrucao

Estrutura de seleção IF

```c
#include<stdio.h>
int main(){
   int x = 10;

   if(x<=10)
      printf("x e menor ou igual a 10");
   else
      printf("x e maior do que 10");
}
```

Estrutura de seleção IF

- Caso tenha uma condição de senão e outro if

```
if(condicao)
  instrucao
else if(condicao)
   instrucao
else
   instrucao
```

Estrutura de seleção IF

```c
#include<stdio.h>
int main(){
  int x = 10;

  if(x<=10)
    printf("x e menor ou igual a 10");
  else if(x<=20)
    printf("x e menor ou igual a 20");
  else
    printf("x e maior do que 20");
  return 0;
}
```

Estrutura de seleção IF

- Caso tenha uma condição de senão e outro if

```
if(condicao){
  instrucoes
}
```