Crash Course C/C++

Roberto Parra

C/C++

Aula 2

Palavras-Chave
1° semestre de 2004

C - 14 Palavras

- for do while (iteração)

C - 14 Palavras

- for do while (iteração)
- if else (decisão)

C - 14 Palavras

- for do while (iteração)
- if else (decisão)
- switch case break default (múltipla escolha)

C - 14 Palavras

- for do while (iteração)
- if else (decisão)
- switch case break default (múltipla escolha)
- typedef enum struct union (estrutura de dado

C - 14 Palavras

- for do while (iteração)
- if else (decisão)
- switch case break default (múltipla escolha)
- typedef enum struct union (estrutura de dado
- return

Maldição

Tá bom Tá bom, tem também o 'goto'

Mas lembrem-se: Quem desse comando se utilizar, há de sofrer inenarráveis tormentos no profundo poço da desgraça eterna.

Laços

FOR

bloco 1 (inicializa)

ponto de retorno

Testa Condição : Falso ==> Termina

Executa Bloco Funcional

bloco 3 (incrementa)

vá para o ponto de retorno

WHILE

ponto de retorno

testa condição : Falso ==> Termina

executa bloco funcional

vá para o bloco funcional

Laços - Exemplos

```
while( CONDICAO )
{
    faca_algo_com( CONDICAO );
}
```

```
int idx=0;
...
while( idx++ < 10 )
{
     matrix[idx] = idx * idx;
}
```

```
int idx=0;
...
do
{
     matrix[idx] = idx * idx;
} while( idx++ < 10 );
```

```
#define ever (;;)
#define young {printf("sem noção ");}
......
for ever young;
```

```
for( index=0; index<10; index++ )
{
    matrix[index] = index * index;
}
```

condicionais

```
if( CONDICAO ) to_ai();
else
{
    to_nem_ai();
    to_nem_aqui();
    raios_onde_estou();
}
```

```
if( zIndex < zLimite )
{
     matrix[idx] = idx * idx;
 zIndex += 2;
}
else if( zIndex > 5 )
{
     zIndex = 0;
}
else // pegadinha
{
    matrix[ aflicmorum ] = 0;
    aflicmorum = pontes( albeit );
}
```

múltipla escolha

```
char cOption;

switch( cOption )
{
   case 'a':
       aha();
   break;
   case 'e':
       hehe();
   break;
   case 'o':
       ohhh();
   break;
   case 'u':
       uuuh();
   break;
   default:
       muhohahah();
}
```

enquanto isso, no lustre do castelo...

```
if( 'a' == cOption ) { aha(); }
else if( 'e' == cOption ) { hehe(); }
else if( 'o' == cOption ) { ohhh(); }
else if( 'u' == cOption ) { uuuh(); }
else { muhohahah(); }
```

(Lana fez em FORTRAN...)

```
IF ( I < 5 ) GOTO ( 1,5,10,30,40 ) I

1 WRITE(UNIT=*,FMT=*) 'UM'
....
40 WRITE(UNIT=2,FMT='A15 I7') A,B,FN
```

funções

```
void hahaha()
{
   printf( "hahaha\t" );
}

void hahaha( void )
{
   printf( "hahaha\t" );
}
```

```
int egral( void )
{
   return 1;
}

int main( int argc, char** argv )
{
   return 0;
}
```

funções

Uma função só transporta argumentos canônicos, ou seja dos tipos: int, float double, etc

```
void vazio
(
    int Eiro,
    float Er,
    double U
)
{
   // retorno tipo 'vazio',
   // mas recebo quaisquer
   // variáveis canônicas:
   // jamais receberei,
   // um vetor como argumento !
   // e sendo 'void',
   // dispenso 'return'
}
```

```
int main( int argc, char** argv )
{
   // enquanto argv[i] é um vetor
   // *argv[i] é um pointer e
   // portanto ocupa apenas
   // sizeof(int) == 4 bytes
   // que é um inteiro e
   // portanto uma variável
   // canônica.
   //
   // argv é um pointer para
   // um vetor de pointers
   // para vetores de chars

   return 0;
}
```

Estruturas de Dados

```
struct __congelador
{
   float rTemperatura;
   float rVolume;
};
struct __torneira
{
   float rVazao;
   float rTemperatura;
};
struct __prateleira
{
   int zMesh;
   float rArea;
};
```

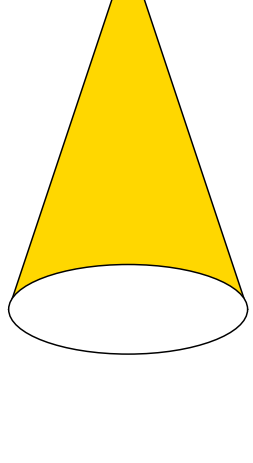
```
typedef struct __prateleira _prateleira;
typedef struct __torneira   _torneira;
typedef struct __congelador _congelador;

struct _geladeira
{
   _congelador congelador;
   _torneira   torneira;
   _prateleira prateleiras[5];
};

typedef struct __geladeira _geladeira;
```

Conceito Typedef

DEFINIÇÃO DA ESTRUTURA DE DADOS

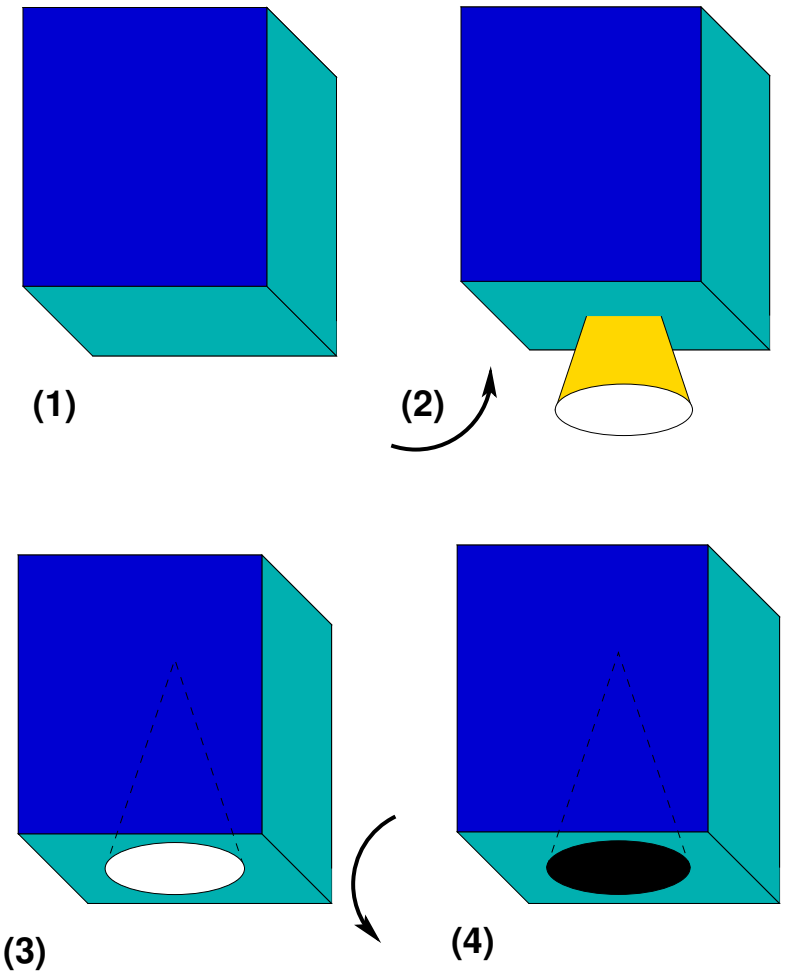
o cone ao lado simboliza a definição da estrutura.

"__cone"é o nome da estrutura:



```
struct __cone
{
   int cor_lateral;
   int cor_base;
   float altura;
   float raio;
};
```

Conceito Typedef

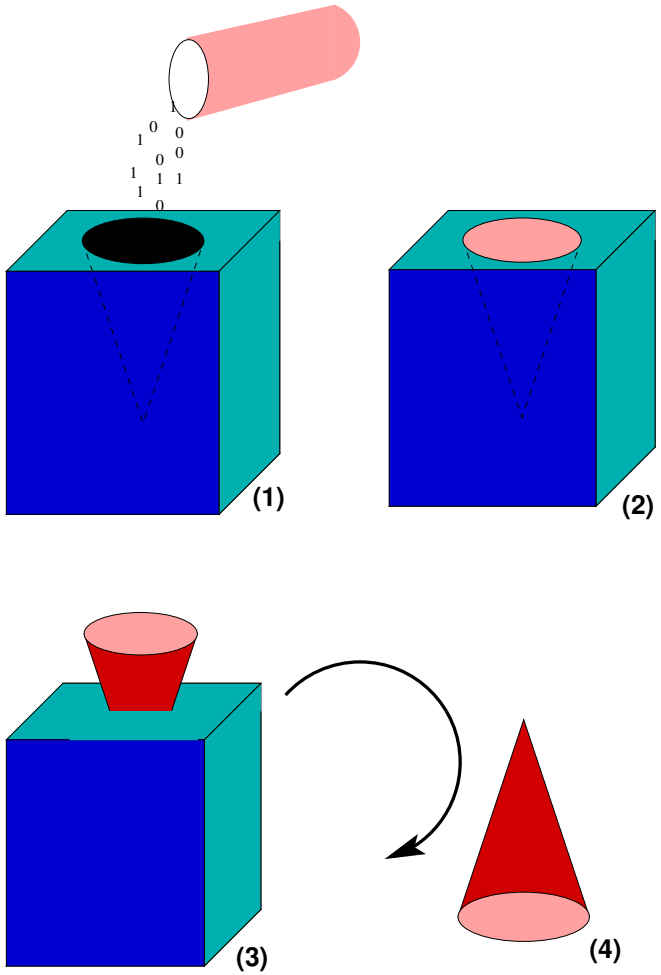


Operador TYPEDEF

(1) o operador typedef é um bloco potencialmente maleável para conter qq forma de dados
(2) **baseado** na estrutura __cone,
(3) crio um **molde** da estrutura
(4) a esse molde dou o nome de **_cone**

```
typedef struct __cone _cone;
```

Conceito Typedef



INSTANCIANDO CONES

(1) usando o molde **_cone**,
(2) "encho-o"de memória,
(3) criando uma estrutura
(4) que pode ser utilizada agora no programa. essa estrutura, cujo nome é **cone**, existe na memória tendo endereço específico de acesso.

```
int main( void )
{
   int contador;
   float area;
   _cone cone;

   .....
}
```

Estruturas de Dados

```
int main( void )
{
   _geladeira frigidaire;

   printf
   (
      "Magna: %d\n",
      sizeof(frigidaire)
   );
}
```

a convenção utilizada aqui para os nomes visa somente separar qualitativamente um especificador de outro. Note que __congelador (dois underscores) é o **nome de uma estrutura de dados** enquanto que _congelador (um underscore) foi definido como um **tipo de variável** (como int,float,etc).

Note então que **congelador** (sem underscore) é uma **variável** do **tipo _congelador** que é definido a partir da **estrutura __congelador**. A variável frigidaire seguiu o mesmo caminho, tendo sido criada pelo tipo _geladeira, que foi definido a partir da estrutura __geladeira.

Estruturas de Dados

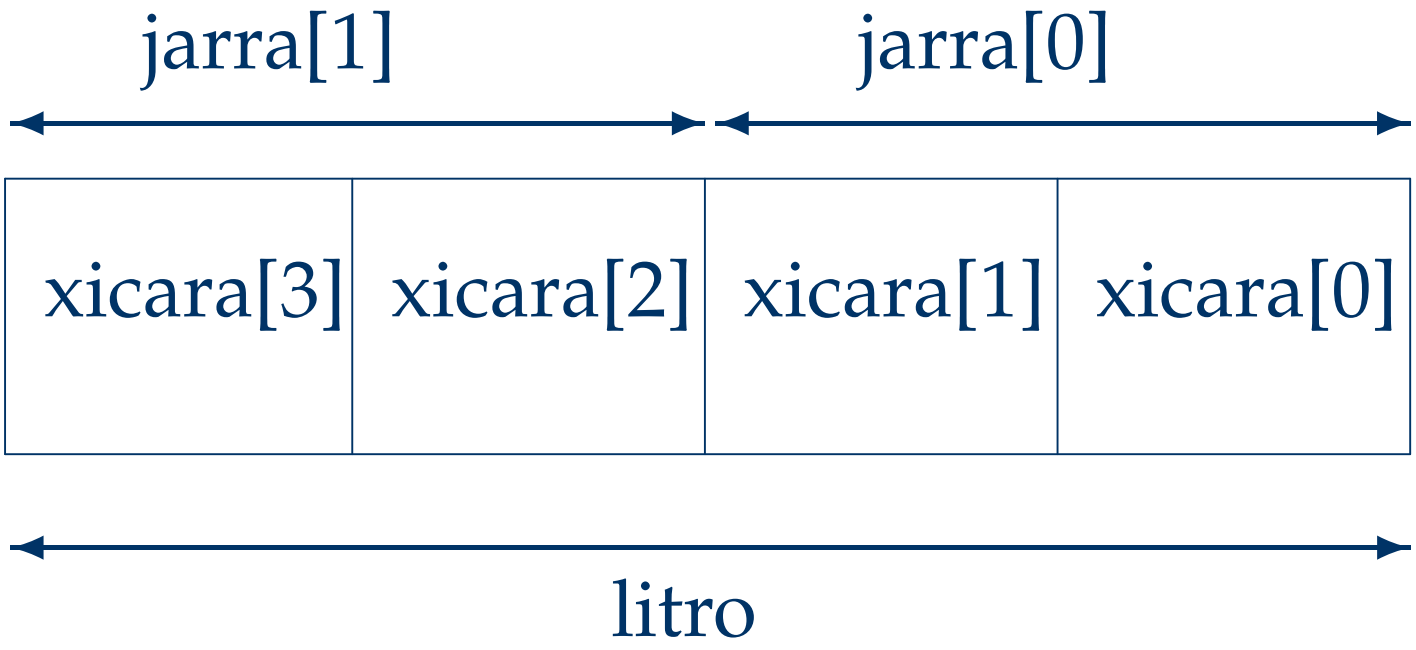
```
union copo_de_liquidificador
{
   long int   litro;
   short int  jarra[2];
   char       xicara[4];
};
```

Uma **union** cria nomes diferentes e identificadores diferentes para a mesma região na memória. A vantagem está em acessar uma área de memória que pode ser gravada sob um tipo de variável e lida sob outro tipo.

Neste caso, podemos gravar um valor usando long (cuba.word) e ler cada grupo de 2 bytes dessa variável (acessando por exemplo verbo.halfword[0]) ou ainda ler byte por byte do long int armazenado (acessando por exemplo verbo.byte[3]).

Estruturas de Dados

union copo_de_liquidificador



Estruturas de Dados

```
union cuba
{
   _pia       pia;
   _tanque    tanque;
   _lavatorio lavatorio;
};
```

Aqui partilham a mesma área de memória três variáveis (pia, tanque e lavatório) sendo que são três tipos diferentes (_pia, _tanque e _lavatorio). O tamanho destra estrutura de dados é a menor boundary do sistema em questão que é maior que o tamanho do maior constituinte.

Boundaries

- Máqs de 16 bits 'andam' de 2 em 2 bytes
- Máqs de 32 bits 'andam' de 4 em 4 bytes
- Máqs de 64 bits 'andam' de 8 em 8 bytes

Boundaries

1 byte	char	auto	register
2 bytes	short int	volatile	static
4 bytes	long int	signed	unsigned
8 bytes	long long int		

float
double
long double

declarações

declarando: **int contador;**

na verdade você está declarando:

auto volatile signed long int contador;

Finalmentes

Roberto Parra nUSP 1694869

esta apresentação foi totalmente elaborada em **LaTeX**, no **slackware** linux ftrn

- 100% Free Software
- 100% MS free !
- 100% elétrons recicláveis

e lembrem-se: Heisenberg *pode* ter estado aqui !