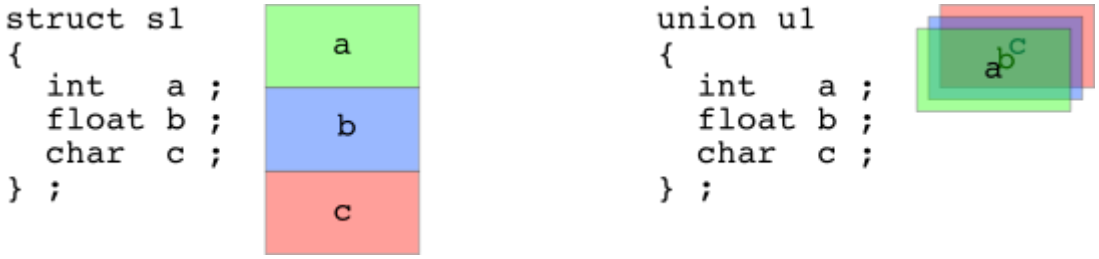
Uniões

Video desta aula

Uma **união** (`union`) é um tipo especial de estrutura (`struct`) no qual todos os campos internos são sobrepostos e ocupam a mesma posição na memória. Com isso, uma mesma posição na memória pode ser "vista" de diversas formas, conforme o campo da união que for acessado.

A figura a seguir ilustra a diferença entre `struct` e `union`:



Uniões são uma forma eficiente de armazenar valores de diferentes tipos em uma mesma posição de memória. Obviamente, somente um valor pode ser armazenado a cada instante. A quantidade de memória ocupada por uma união corresponde ao tamanho de seu **maior campo**.

No exemplo abaixo, uma união permite "enxergar" os bytes individuais de um inteiro sem necessidade de operações de bits ou aritmética de ponteiros:

union.c

```c
#include <stdio.h>

// guarda um inteiro OU um vetor de 4 bytes
typedef union
{
  int value ;
  unsigned char byte[sizeof(int)] ;
} intParts ;

int main ()
{
  intParts a ;
  int i ;

  a.value = 653459 ;

  printf ("%d: ", a.value) ;

  for (i = 0; i < sizeof (int); i++)
    printf ("%02x ", a.byte[i]) ;
  printf ("\n") ;
}
```

Os campos `a.value` e `a.byte[]` estão alocados da seguinte forma na memória:

addr	***addr+1***	***addr+2***	***addr+3***
a.value (4 bytes)			
a.byte[0]	a.byte[1]	a.byte[2]	a.byte[3]

Uniões podem ser usadas para armazenar valores de diversos tipos em uma única locação de memória. Por exemplo, a união a seguir pode ser usada para armazenar números de diversos tipos:

```
typedef union
{
  short  shortVal ;
  int    intVal ;
  long   longVal ;
  float  floatVal ;
  double doubleVal ;
} numeric_t ;
```

Entretanto, como saber qual o tipo do último valor armazenado, ou seja, qual o valor corrente?

Se armazenarmos um `int` e tentarmos ler um `float` teremos um valor errado, pois os valores são lidos byte a byte diretamente da área de memória da união, sem conversões.

union-error.c

```
#include <stdio.h>

typedef union
{
  short  shortVal ;
  int    intVal ;
  long   longVal ;
  float  floatVal ;
  double doubleVal ;
} numeric_t ;

int main ()
{
  numeric_t a ;

  a.shortVal = 741 ;
  printf ("short : %d\n", a.shortVal) ;
  printf ("float : %f\n\n", a.floatVal) ;

  a.floatVal = 327.5432 ;
  printf ("float : %f\n", a.floatVal) ;
  printf ("short : %d\n\n", a.shortVal) ;

  a.doubleVal = 327.5432 ;
  printf ("double: %lf\n", a.doubleVal) ;
  printf ("float : %f\n", a.floatVal) ;
  printf ("short : %d\n", a.shortVal) ;
  return 0 ;
}
```

Para resolver esse problema pode ser usada uma estrutura contendo a união e uma variável que indique o tipo do último valor armazenado:

union-type.c

```c
#include <stdio.h>

typedef struct
{
  union        // ATTENTION: "anonymous" union
  {
    short  shortVal ;
    int    intVal ;
    long   longVal ;
    float  floatVal ;
    double doubleVal ;
  } ;
  enum { SHORT, INT, LONG, FLOAT, DOUBLE } type ;
} numeric_t ;

// imprime tipo numérico
void print_num (numeric_t n)
{
  switch (n.type)
  {
    case SHORT  : printf ("%d", n.shortVal)   ; break ;
    case INT    : printf ("%d", n.intVal)     ; break ;
    case LONG   : printf ("%ld", n.longVal)   ; break ;
    case FLOAT  : printf ("%f", n.floatVal)   ; break ;
    case DOUBLE : printf ("%lf", n.doubleVal) ; break ;
    default     : printf ("NaN") ;
  }
}

int main ()
{
  numeric_t a ;

  a.shortVal = 119 ;
  a.type = SHORT ;
  print_num (a) ;
  printf ("\n") ;

  a.longVal = 3451212796756 ;
  a.type = LONG ;
  print_num (a) ;
  printf ("\n") ;

  a.doubleVal = 3.141592653589793 ;
  a.type = DOUBLE ;
  print_num (a) ;
  printf ("\n") ;
}
```

O exemplo acima traz um exemplo de **união anônima**, ou seja, sem nome. Nesse caso, os membros da união são considerados como membros da estrutura externa que contém a união. Estruturas também podem ser anônimas.

A linguagem C respeita a ordem de declaração dos campos de uma estrutura, ou seja, eles são colocados na memória na mesma ordem em que são declarados.

Isso permite outro exemplo interessante do uso de uma união, no qual as moedas podem ser acessadas com nomes individuais ou como elementos de um vetor:

```
typedef union {
  struct
  {
    int quarter;
    int dime;
    int nickel;
    int penny;
  };
  int coins[4];
} Coins_t ;

Coins_t a ;

a.dime = 34 ;          // são operações equivalentes
a.coins[1] = 34 ;
```

Outro exemplo interessante é a definição de números reais segundo o padrão IEEE 754. Ele permite acessar o valor real ou suas partes (mantissa, expoente e sinal) separadamente:

```
// extraído (e simplificado) de /usr/include/x86_64-linux-gnu/ieee754.h

union ieee754_float
{
  float f;

  /* This is the IEEE 754 single-precision format.  */
  struct
  {
    #if __BYTE_ORDER == __BIG_ENDIAN
    unsigned int negative:1;
    unsigned int exponent:8;
    unsigned int mantissa:23;
    #endif                          /* Big endian.  */
    #if __BYTE_ORDER == __LITTLE_ENDIAN
    unsigned int mantissa:23;
    unsigned int exponent:8;
    unsigned int negative:1;
    #endif                          /* Little endian.  */
  } ieee;
};
```

Exercícios

1. Utilizando uma única variável `union`, crie uma função que receba um inteiro e calcule seu quadrado, em seguida, receba um caractere e, caso maiúsculo, imprima minúsculo, caso minúsculo, imprima maiúsculo, e por último, receba uma string de no máximo 8 caracteres e imprima seu inverso.
2. Variáveis de 32 bits do tipo `int` podem representar valores entre −2,147,483,647 e +2,147,483,647, enquanto variáveis de 32 bits do tipo `unsigned int` podem representar valores entre 0 e +4,294,967,295. Crie um programa que receba um valor negativo do tipo `int` e mostre qual o valor resultante da conversão para o tipo `unsigned int`.
3. Utilizando unions, crie um programa capaz de receber um determinado valor e calcular o módulo de 256 desse valor (dica: utilize `char[sizeof(int)]`).