Linguagem C

‘Strings’

Inicialização

- Podemos utilizar arrays para representar ‘strings’, tal como outros arrays devemos declarar com um tamanho fixo:

```
char text[100]; //100 caracteres no máximo
```

- As ‘strings’ em C são terminadas com um carácter de código zero (`\0`)
    - a ‘string’ “Abba” é
    - a ‘string’ vazia é
- O carater `\0` é um *terminator* que marca o final da ‘string’
    - não faz parte do texto
- O comprimento da ‘string’ é determinado pela posição do terminador.

Inicialização(cont.)

- Inicializar usando constantes de caracteres:
  `char text[6] = {'H','e','l','l','o','\0'};`
- Podemos também inicializar com os caracteres entre aspas:
  `char text[6] = "Hello";`
    - Desta forma o terminador `\0` é automaticamente introduzido no final
- Podemos declarar um tamanho superior ao necessário:
  `char text[100] = "Hello"; //os caracteres restantes são \0`
- Se omitirmos o tamanho, o compilador reserva apenas o necessário
  `char text[] = "Hello"; //tamanho 6`

Imprimir ‘strings’

- Para imprimir uma ‘string’ podemos usar:
    - a função `puts`
    - ou `printf` com o formato `%s`

```
#include<stdio.h>
int main(...){
  char text[] = "Hello, world";
  puts(text);
}
```

- Alternativa:

```
#include<stdio.h>
int main(...){
  char text[] = "Hello, world";
  printf("%s\n", text);
}
```

Ler ‘strings’

- Podemos usar o `fgets()` para ler strings:

```
#include<stdio.h>
#define MAX_SIZE 100
...
char text[MAX_SIZE];
fgets(text, MAX_SIZE,stdin);
```

  - o segundo argumento especifica o tamanho máximo
  - o terceiro argumento especifica o canal de entrada (`stdin` é a entrada padrão)

Processar 'strings'

Contar comprimento

- Uma função para calcular o *comprimento* duma string:
  `unsigned comprimento(char str[]);`
- Existe uma função semelhante na biblioteca-padrão (`string.h`)
  `size_t strlen(char str[])`
- `size_t` é um inteiro sem sinal (porque o comprimento nunca é negativo)

Exercício
Implementar a função `unsigned comprimento(char str[]);`

Contar espaços

- Definamos uma função que contém o *número de espaços* numa 'string'
  `unsigned contar_espacos(char str[]);`
- Não necessitamos de passar o comprimento como argumento porque o fim da cadeia é marcado por `\0`

Exercício

Implementar a função `unsigned contar_espacos(char str[]);`

Inverter uma string

- Vamos definir uma função para inverter a ordem duma string
  - "abc123" deve ser transformado em "321cba"
- Vamos modificar a string passada como argumento:
  `void inverter(char str[]);`
- A função `inverter` não retorna qualquer valor

Exercício

Implementar a função `void inverter (char str[])`

Procurar um carater

```
int procurar(char str[], char ch);
```

- Procura a *primeira ocorrência* do carater ch na 'string'
- Se encontrar, retorna o seu índice;
- Caso contrário, retorna um índice inválido (-1)

Exercício

Implementar a função `int procurar(char str[], char ch)`

Comparar strings

```
int comparar(char str1[], char str2[])
```

- Testar se duas strings contém os mesmos carateres pela mesma ordem;
- Resultado: 1 em case afirmativo, 0 em caso contrário
- Exemplo:

```
char texto1[] = "ABC";
char texto2[] = "ABC123";
c = comparar(texto1, texto2);
 // resultado 0 (diferentes
```

- Comparar com == **não** produz o efeito desejado: compara os *endereços* das strings e não o conteúdo

Exercício

Implementar a função `int comparar(char str1[], char str2[])`

Imprimir uma linha

```
void imprimir(char str[]);
```

- Imprimir todos os carateres seguidos de `\n`
- Semelhante à função `puts` da biblioteca padrão
- Usar `putchar` para imprimir um carater de cada vez.

Exercício

Implementar a função `void imprimir(char str[]);`

Ler uma linha

```
int ler_linha(char str[], int max_size);
```

- Ler carateres da entrada padrão até `\n` ou `EOF`
- Guardar na cadeia `str` e terminar com `\0`
- Ler no máximo `max_size-1` carateres (devido ao terminador)
- Retorna o número de carateres guardado.

Exercício

Implementar a função `int ler_linha(char str[], int max_size);`

Apontadores

Arquitetura de memória

- Nos computadores atuais a memória é organizada em *palavras* de tamanho fixo
- Tipicamente cada palavra tem um *byte*

- Cada palavra tem um *endereço* único
- O processador pode aceder a cada palavra individualmente pelo seu endereço
- Podemos ver os endereços duma meméroa com n palavras como sendo os inteiros de 0 até n-1
- Exemplo: 64Kb de memória(=64 x 1024 bytes)

endereço	conteúdo
0	10101010
1	11111010

65535	11010010

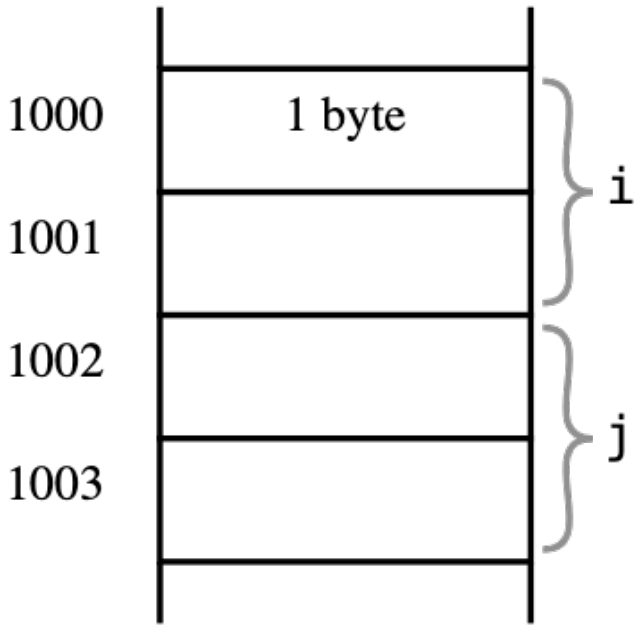
Arquitetura de memória(cont.)

- Como os tamanhos de memórias são normalmente potências de 2 é conveniente usar **notação hexadecimal** (base 16) para os endereços
- Exemplo: os endereços duma memória de 64Kb vão de 0x0000 até 0xffff

Variáveis

- Em C cada variável ocupa uma ou mais palavras em memória
- O *endereço da variável* é o endereço da primeira palavra

Exemplo



- i, j são inteiros
- supomos que cada inteiro ocupa 2 bytes
- o endereço de i é 1000 e o de j é 1002

Operador &

- O operador &obtém o **endereço** duma variável
- Podemos imprimir usando `printf`com formato %p: