Programação Estruturada

Aula 17 - Strings

Yuri Malheiros (yuri@ci.ufpb.br)

Manipulando Strings

- Programadores C costumam ter um estilo particular para manipular strings
- Aprendemos muitos entendendo como esse estilo funciona
- Ler código de qualidade é um bom exercício!

Procurando pelo fim de um string

- Muitas operações precisam varrer um string até o seu fim
  - `strlen` é um exemplo
- Vamos implementar uma versão nossa do `strlen`

Procurando pelo fim de um string

```
unsigned int strlen(const char *s) {
  unsigned int n;

  for (n=0; *s!='\0'; s++)
    n++

  return n;
}
```

Procurando pelo fim de um string

- Uma versão mais condensada:

```
unsigned int strlen(const char *s) {
  unsigned int n=0;

  for (; *s++;)
    n++

  return n;
}
```

Procurando pelo fim de um string

- Uma versão mais condensada com while:

```
unsigned int my_strlen(const char *s) {
  unsigned int n=0;

  while (*s++)
    n++;

  return n;
}
```

Copiando um string

- Copiar um string é outra operação comum
- Vamos desenvolver duas versões de `strcat`

Copiando um string

```
char *my_strcat(char *s1, const char *s2) {
  char *p = s1;

  while (*p != '\0')
    p++;

  while (*s2 != '\0') {
    *p = *s2;
    p++;
    s2++;
  }

  *p = '\0';

  return s1;
}
```

Copiando um string

- Essa versão do `strcat` usa um algoritmo com duas etapas:
  - Encontrar o fim do primeiro string e fazer `p` apontar pra ele
  - Copiar caracteres de `s2` para onde `p` estiver apontando

Copiando um string

- Uma versão mais condensada:

```
char *my_strcat(char *s1, const char *s2) {
  char *p = s1;

  while (*p)
    p++;

  while (*p++ = *s2++)
    ;

  return s1;
}
```

Arrays de Strings

- Podemos armazenar um array de strings usando um array de caracteres bidimensional

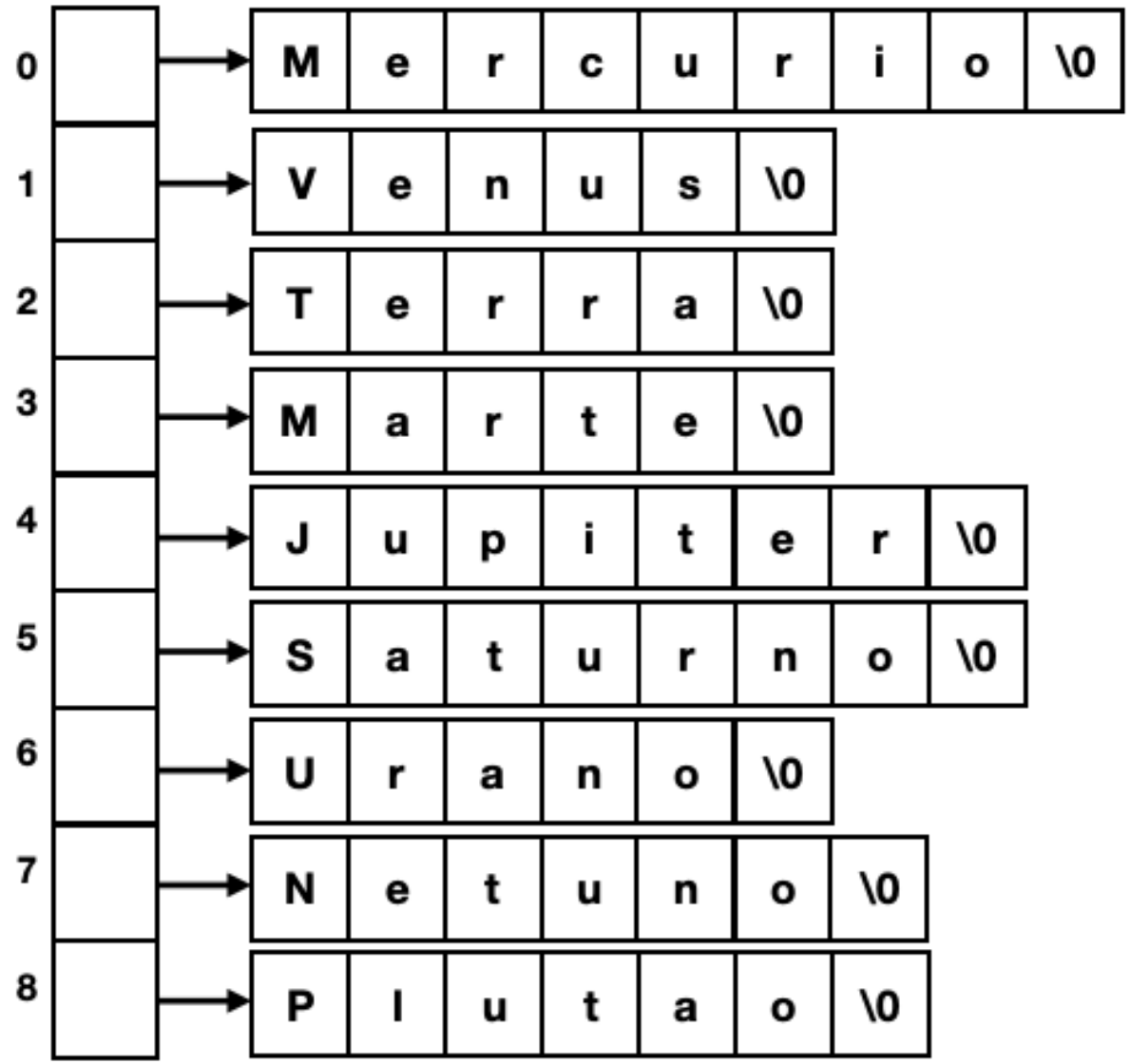
```
char planets[][9] = {"Mercurio", "Venus", "Terra",
                     "Marte", "Jupiter", "Saturno",
                     "Urano", "Netuno", "Plutao"}
```

	0	1	2	3	4	5	6	7	8
0	M	e	r	c	u	r	i	o	\0
1	V	e	n	u	s	\0	\0	\0	\0
2	T	e	r	r	a	\0	\0	\0	\0
3	M	a	r	t	e	\0	\0	\0	\0
4	J	u	p	i	t	e	r	\0	\0
5	S	a	t	u	r	n	o	\0	\0
6	U	r	a	n	o	\0	\0	\0	\0
7	N	e	t	u	n	o	\0	\0	\0
8	P	l	u	t	a	o	\0	\0	\0

Arrays de Strings

- Essa representação é ineficiente
- Vários strings não ocupam o espaço reservado para eles
- É mais eficiente armazenar esse array como um array de ponteiros para strings

```
char *planets[] = {"Mercurio", "Venus", "Terra",
                   "Marte", "Jupiter", "Saturno",
                   "Urano", "Netuno", "Plutao"}
```


0
M e r c u r i o \0
1
V e n u s \0
2
T e r r a \0
3
M a r t e \0
4
J u p i t e r \0
5
S a t u r n o \0
6
U r a n o \0
7
N e t u n o \0
8
P l u t a o \0

Arrays de Strings

- Cada elemento do array `planets` é um ponteiro para um string que termina em `'\0'`
- Não existe desperdício de espaço
- Para acessar um string, basta utilizar a indexação
    - `planets[0]`

Argumentos de linha de comando

- Quando executamos programas pela linha de comando é comum precisar passar alguma informação para ele
- `ls` - lista os arquivos de um diretório
- `ls -l` - exibe uma lista mais detalhada dos arquivos do diretório
- `ls -l arquivo.c` - exibe os detalhes só do arquivo.c

Argumentos de linha de comando

- Os programas em C podem receber informações passadas pela linha de comando
- Para isso precisamos modificar o `main`

```
int main (int argc, char *argv[]) {
  ...
}
```

- `argc` - argument count - é o número de argumentos (incluindo o nome do próprio programa) passados pela linha de comando
- `argv` - argument vector - é um array de strings com os argumentos passados na execução do programa

Argumentos de linha de comando

- Para o `ls` se o usuário digitar `ls -l arquivo.c`
- `argc` vai ser 3
- `argv[0]` vai ser `ls`
- `argv[1]` vai ser `-l`
- `argv[0]` vai ser `arquivo.c`

Argumentos de linha de comando

- Podemos checar os argumentos passados com o código:

```
for (int i=0; i<argc; i++)
  printf("%s\n", argv[i]);
```