Programação
de computadores

Pedro Melo-Pinto
CITAB - Universidade de Trás-os-Montes e Alto Douro
Quinta de Prados – Apartado 1013. 5000-811 Vila Real - PORTUGAL
pmelo@utad.pt

2018-2019

strings: utilização e manipulação

14

strings. conceitos

As *strings* são *arrays* (cadeias) de caracteres alfanuméricos

Uma *string* é designada por um identificador.
exemplo:
char str[9] = {"Paulo"};

P	a	u	l	o	\0	?	?	?
0	1	2	3	4	5	6	7	8

vector str (de caracteres alfanuméricos)

Em linguagem C, por convenção, as *strings* devem terminar com o caracter especial '\0'.
A não colocação desse último caracter pode gerar erros no decorrer do programa, pois estaríamos na presença de uma *string* "infinita".

strings. conceitos

As *strings* são *arrays* (cadeias) de caracteres alfanuméricos

Uma *string* é designada por um identificador.
exemplo:
char str[9] = {"Paulo"};

P	a	u	l	o	\0	?	?	?
0	1	2	3	4	5	6	7	8

vector str (de caracteres alfanuméricos)

As *strings* são guardadas utilizando o código ASCII

strings. conceitos

Código ASCII (American Standard Code for Information Interchange)

Hex	Dec	Char		Hex	Dec	Char		Hex	Dec	Char		Hex	Dec	Char	
0x00	0	NULL	null	0x20	32	Space		0x40	64	@		0x60	96	`	
0x01	1	SOH	Start of heading	0x21	33	!		0x41	65	A		0x61	97	a	
0x02	2	STX	Start of text	0x22	34	"		0x42	66	B		0x62	98	b	
0x03	3	ETX	End of text	0x23	35	#		0x43	67	C		0x63	99	c	
0x04	4	EOT	End of transmission	0x24	36	$		0x44	68	D		0x64	100	d	
0x05	5	ENQ	Enquiry	0x25	37	%		0x45	69	E		0x65	101	e	
0x06	6	ACK	Acknowledge	0x26	38	&		0x46	70	F		0x66	102	f	
0x07	7	BELL	Bell	0x27	39	'		0x47	71	G		0x67	103	g	
0x08	8	BS	Backspace	0x28	40	(		0x48	72	H		0x68	104	h	
0x09	9	TAB	Horizontal tab	0x29	41	)		0x49	73	I		0x69	105	i	
0x0A	10	LF	New line	0x2A	42	*		0x4A	74	J		0x6A	106	j	
0x0B	11	VT	Vertical tab	0x2B	43	+		0x4B	75	K		0x6B	107	k	
0x0C	12	FF	Form Feed	0x2C	44	,		0x4C	76	L		0x6C	108	l	
0x0D	13	CR	Carriage return	0x2D	45	-		0x4D	77	M		0x6D	109	m	
0x0E	14	SO	Shift out	0x2E	46	.		0x4E	78	N		0x6E	110	n	
0x0F	15	SI	Shift in	0x2F	47	/		0x4F	79	O		0x6F	111	o	
0x10	16	DLE	Data link escape	0x30	48	0		0x50	80	P		0x70	112	p	
0x11	17	DC1	Device control 1	0x31	49	1		0x51	81	Q		0x71	113	q	
0x12	18	DC2	Device control 2	0x32	50	2		0x52	82	R		0x72	114	r	
0x13	19	DC3	Device control 3	0x33	51	3		0x53	83	S		0x73	115	s	
0x14	20	DC4	Device control 4	0x34	52	4		0x54	84	T		0x74	116	t	
0x15	21	NAK	Negative ack	0x35	53	5		0x55	85	U		0x75	117	u	
0x16	22	SYN	Synchronous idle	0x36	54	6		0x56	86	V		0x76	118	v	
0x17	23	ETB	End transmission block	0x37	55	7		0x57	87	W		0x77	119	w	
0x18	24	CAN	Cancel	0x38	56	8		0x58	88	X		0x78	120	x	
0x19	25	EM	End of medium	0x39	57	9		0x59	89	Y		0x79	121	y	
0x1A	26	SUB	Substitute	0x3A	58	:		0x5A	90	Z		0x7A	122	z	
0x1B	27	FSC	Escape	0x3B	59	;		0x5B	91	[		0x7B	123	{	
0x1C	28	FS	File separator	0x3C	60	<		0x5C	92	\		0x7C	124	|	
0x1D	29	GS	Group separator	0x3D	61	=		0x5D	93	]		0x7D	125	}	
0x1E	30	RS	Record separator	0x3E	62	>		0x5E	94	^		0x7E	126	~	
0x1F	31	US	Unit separator	0x3F	63	?		0x5F	95	_		0x7F	127	DEL	

strings. conceitos

As *strings* são *arrays* (cadeias) de caracteres alfanuméricos

Uma *string* é designada por um identificador.
exemplo:
char str[9] = {"Paulo"};

P	a	u	l	o	\0	?	?	?
0	1	2	3	4	5	6	7	8

vector str (de caracteres alfanuméricos)

As *strings* são guardadas utilizando o código ASCIII

80	97	117	108	11	0	?	?	?
0	1	2	3	4	5	6	7	8

vector str (codificação ASCII)

Codificação (ASCII)

Hex	Dec	Char		Hex	Dec	Char
0x41	65	A		0x61	97	a
0x4C	76	L		0x6C	108	l
0x4D	77	M		0x6D	109	m
0x4E	78	N		0x6E	110	n
0x4F	79	O		0x6F	111	o
0x50	80	P		0x70	112	p
0x51	81	Q		0x71	113	q
0x52	82	R		0x72	114	r
0x53	83	S		0x73	115	s
0x54	84	T		0x74	116	t
0x55	85	U		0x75	117	u

strings. conceitos

As *strings* são *arrays* (cadeias) de caracteres alfanuméricos

Valores literais (constantes)
Os valores literais de strings são representados por sequências de caracteres entre aspas (“).

Exemplos
“” : string vazia
“Paulo”

“a” **vs** ‘a’:

‘a’ é um caracter simples (tipo de dados primitivo char), cuja representação através do seu código ASCII é guardada num byte.

“a” é um vector de dois caracteres, sendo o primeiro a, e o segundo o delimitador de fim de *string* \0.

strings. conceitos

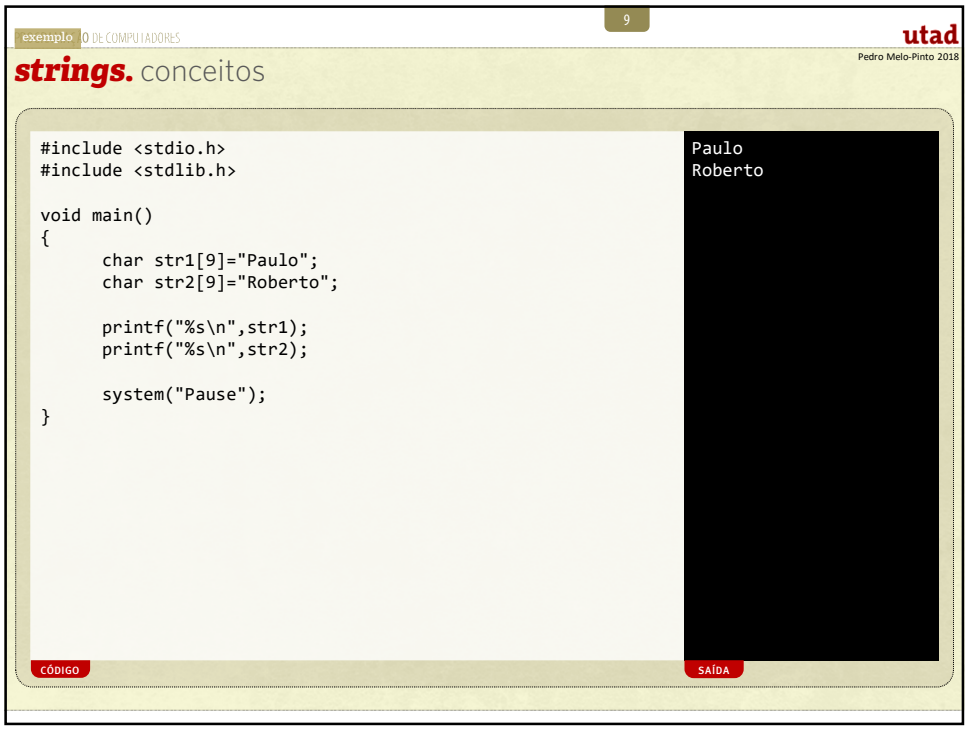
As *strings* são *arrays* (cadeias) de caracteres alfanuméricos

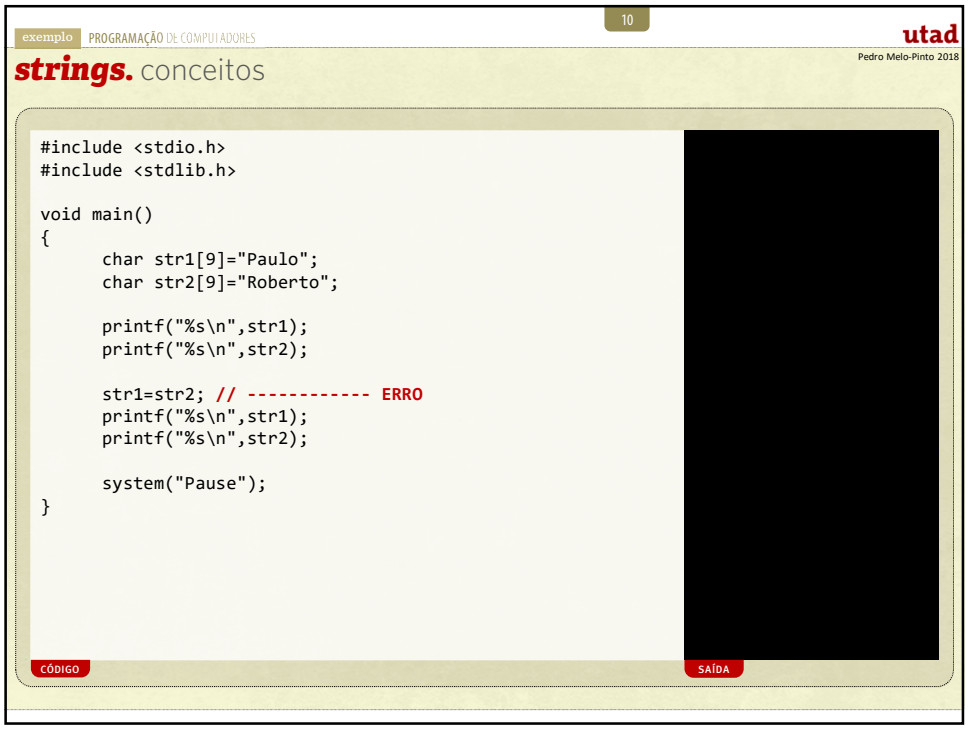
Variáveis de tipo *array* de caracteres alfanuméricos – *strings*
Declarar um vector de tamanho suficiente para conter a *string* (+1 para o delimitador \0)

Exemplos (com inicialização incluída)

```
char str1[6] = “Paulo”;
char str2[] = “Paulo”;
char *str3 = “Paulo”;
char str4[6] = {‘P’,’a’,’u’,’l’,’o’,’\0’};

str1 = str2;  // não é permitido
```


9
exemplo 0 DE COMPUTADORES
utad
Pedro Melo-Pinto 2018
strings. conceitos
#include <stdio.h>
#include <stdlib.h>
void main()
{
char str1[9]="Paulo";
char str2[9]="Roberto";
printf("%s\n",str1);
printf("%s\n",str2);
system("Pause");
}
Paulo
Roberto
CÓDIGO
SAÍDA


10
exemplo PROGRAMAÇÃO DE COMPUTADORES
utad
Pedro Melo-Pinto 2018
strings. conceitos
#include <stdio.h>
#include <stdlib.h>
void main()
{
char str1[9]="Paulo";
char str2[9]="Roberto";
printf("%s\n",str1);
printf("%s\n",str2);
str1=str2; // ------------ ERRO
printf("%s\n",str1);
printf("%s\n",str2);
system("Pause");
}
CÓDIGO
SAÍDA

strings. conceitos

As *strings* são *arrays* (cadeias) de caracteres alfanuméricos

Variáveis de tipo *array* de caracteres alfanuméricos – *strings*

Declarar um vector de tamanho suficiente para conter a *string* (+1 para o delimitador \0)

Exemplos (com inicialização incluída)

```
char str1[6] = “Paulo”;
char str2[] = “Paulo”;
char *str3 = “Paulo”;
char str4[6] = {‘P’,’a’,’u’,’l’,’o’,’\0’};
```

```
str1 = str2;  // não é permitido
```

As *strings* enquanto vectores não podem ser alteradas por afectação directa, mas podemos utilizar variáveis de tipo apontador para o fazer.

strings. conceitos

CÓDIGO

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
      char *str1="Paulo";
      char *str2="Roberto";

      printf("%s\n",str1);
      printf("%s\n",str2);

      str1=str2;

      printf("\n%s\n",str1);
      printf("%s\n",str2);

      system("Pause");
}
```

SAÍDA

```
Paulo
Roberto

Roberto
Roberto
```

strings. conceitos

As *strings* são *arrays* (cadeias) de caracteres alfanuméricos

Variáveis de tipo *array* de caracteres alfanuméricos – *strings*

As *strings* enquanto vectores não podem ser alteradas por afectação directa, mas podemos utilizar variáveis de tipo apontador para o fazer .

Podemos também alterar parte de uma *string*

```
char str1[6] = "Paulo";
str1[4] = 'a';  // str1 fica com o valor "Paula"
str1[3] = '\0';  // str1 fica com o valor "Pau"
```

Cuidado para não eliminar o delimitador de fim de *string.*
Substituir str1[5] na *string* original com um caracter que não '\0' torna a *string* "infinita".
Respeitar os limites do *array*.

strings. conceitos

CÓDIGO

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
      char str1[9]="Paulo";
      printf("%s\n",str1);

      str1[4]='a';
      printf("%s\n",str1);

      str1[3]='\0';
      printf("%s\n",str1);

      system("Pause");
}
```

SAÍDA

```
Paulo
Paula
Pau
```

strings. conceitos

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
      char str1[6]="Paulo";
      printf("%s\n",str1);

      str1[8]='X'; //--------- ESCREVEU À FRENTE DE str1
      printf("%s\n",str1);

      str1[5]='X';
      printf("%s\n",str1);

      system("Pause");
}
```

CÓDIGO

```
Paulo
Paulo
Paulo??????
```

SAÍDA

strings. entrada de dados

Entrada de dados em *strings*

Utilização do símbolo de formatação %s com *scanf* para ler uma *string*
* Guarda os caracteres até ao próximo espaço (ou *enter* ou equivalente)
* Em linguagem C é acrescentado o caracter *null* (\0)
* Não é necessário & antes da variável pois um *array* é um apontador

Exemplo:

```
char str1[11];
scanf("%s",str1);
```

Problema: não existe limite para o número de caracteres lido, que pode exceder o tamanho do *array*.

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad

Pedro Melo-Pinto 2018

strings. conceitos

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
      char str1[6];

      scanf("%s",str1);
      printf("\nstr: %s\n",str1);

      system("Pause");
}
```

CÓDIGO

```
(teclado:)
Paulo

str: Paulo
----------------------
(teclado:)
  Paulo

str: Paulo
----------------------
(teclado:)
Paulo Gomes

str: Paulo
```

SAÍDA

PROGRAMAÇÃO DE COMPUTADORES

utad

Pedro Melo-Pinto 2018

strings. entrada de dados

Entrada de dados em *strings*

Utilização do símbolo de formatação %s com *scanf* para ler uma *string*
- Ignora espaços no início
- Guarda os caracteres até ao próximo espaço (ou *enter* ou equivalente)
- Em linguagem C é acrescentado o caracter *null* (\0)
- Não é necessário & antes da variável pois um *array* é um apontador

Exemplo:

```
char str1[11];
scanf("%s",str1);
```

Problema: não existe limite para o número de caracteres lido, que pode exceder o tamanho do *array*.

Podemos especificar o números de caracteres a ler:

```
char str1[11];
scanf("%10s",str1);
```

Não esquecer que é necessário o \0.
As *strings* menores do que o especificado são lidas normalmente.

strings. conceitos

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
      char str1[6];

      scanf("%5s",str1);
      printf("\nstr: %s\n",str1);

      system("Pause");
}
```

CÓDIGO

```
(teclado:)
Paulo

str: Paulo
-----------------------
(teclado:)
Bernardo

str: Berna
```

SAÍDA

strings. entrada de dados

Entrada de dados em *strings*

Podemos ainda especificar quais os caracteres admissíveis
%[ListaChars]

- ListaChars especifica o conjunto de caracteres (designado *scan set*)
- Os caracteres que pertencem a este conjunto são lidos
- A *string* acaba quando encontra um caracter não pertencente ao *scan set*

Nota:
Não ignora espaços no início
Qualquer caracter pode fazer parte do *scan set* excepto]
Utilizando ^ no início nega o conjunto

Exemplos:

```
scanf("%[-+0123456789]",Number);
scanf("%[^\n]",Line);  //   Lê até newline
```

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. entrada de dados

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
      char str1[10];

      scanf("%[+-0123456789]",str1);
      printf("\nstr: %s\n",str1);

      system("Pause");
}
```

CÓDIGO

```
(teclado:)
237

str: 237
-----------------------
(teclado:)
237Paulo22

str: 237
-----------------------
(teclado:)
  237

str: ???
```

SAÍDA

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. entrada de dados

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
      char str1[10];

      scanf("%[^+-0123456789\n]",str1);
      printf("\nstr: %s\n",str1);

      system("Pause");
}
```

CÓDIGO

```
(teclado:)
Paulo

str: Paulo
-----------------------
(teclado:)
Paulo237Gomes

str: Paulo
-----------------------
(teclado:)
   Paulo

str:    Paulo
```

SAÍDA

strings. saída de dados

Saída de dados com *strings*

Para a saída de dados com *strings* utiliza-se o símbolo de formatação % com, por exemplo, a função *printf*:

Exemplos:

Imprimir os caracteres da *string* (até encontrar \0)

```
char str1[10] = "Paulo";
printf("|%s|",str1);   // saída: |Paulo|
printf("%s",str1);   // saída: Paulo
```

Podemos especificar o comprimento a usar na saída:

```
printf("|%10s|",str1);   // saída: |     Paulo|
printf("|%-10s|",str1);   // saída: |Paulo     |
```


strings. saída de dados

Estudo de caso : (16**A**)

Desenhe um programa que leia um nome de uma pessoa, dividido em nome de família e nome próprio (por esta ordem) separados por vígula, e o apresente (nome próprio seguido do nome de família) no ecrã.

Exemplo:

Leitura	Gomes,Paulo
Escrita	Paulo Gomes

exemplo PROGRAMAÇÃO DE COMPUTADORES

strings. saída de dados

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
  char nProprio[11];
  char nFamilia[11];

  printf("Nome (nFamilia,nProprio): ");
  scanf("%[^,],%s",nFamilia,nProprio);

  printf("Bom dia %s %s\n",nProprio,nFamilia);

  system("Pause");
}
```

CÓDIGO

```
(teclado:)
Nome (nFamilia,nProprio): Gomes,Paulo

Bom dia Paulo Gomes
----------------------
(teclado:)
Nome (nFamilia,nProprio): Gomes,  Paulo

Bom dia Paulo Gomes
```

SAÍDA

PROGRAMAÇÃO DE COMPUTADORES

strings. entrada de dados

Entrada de dados em *strings*

Ler linhas completas

char *gets(char *str)
- Lê uma linha (até encontrar *newline*) do teclado e guarda-a no *array* de caracteres str.
- Retorna str se for bem sucedida ou *NULL* se tiver problemas.
- Não tem limite para o número de chars que lê.
- O caracter *newline* está incluído na *string* lida.

char *fgets(char *str, int n, FILE *fp)
- Equivalente a gets() para leitura feita de um ficheiro cujo apontador é fp.
- fp é um apontador para FILE (entrada).
- Lê, no máximo, n caracteres (+1 para \0).
- Retorna str se for bem sucedida ou *NULL* se tiver problemas.
- Para ler do teclado: fgets(mystring,100,stdin)
- O caracter *newline* está incluído na *string* lida.

strings. saída de dados

Saída de dados com *strings*

Imprimir linhas completas

int puts(char *str)
* Envia a *string* str para o ecrã.
* Imprime até encontrar \0.
* Retorna EOF se falhar.
* Envia *newline* se encontrar \n (possível para leituras feitas com gets ou fgets).

int fputs(char *str, FILE *fp)
* Equivalente a gets() para saída feita para um ficheiro cujo apontador é fp.
* fp é um apontador para FILE (saída).
* Retorna EOF se falhar.
* Envia *newline* se encontrar \n.

strings. entrada e saída de dados

Estudo de caso : (16B)

(Utilização de *arrays* de *strings*)
Desenhe um programa que leia do teclado os nomes dos meses do ano, apresentando-os de seguida no ecrã.

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. entrada e saída de dados

```c
#include <stdio.h>
#include <stdlib.h>

void main()
{
  int i;
  char nomeMes[12][10];

  for(i=0; i<12; i++)
  {
     printf("Mes[%2d]: ",i);
     scanf("%9s",nomeMes[i]);
  }

  for(i=0; i<12; i++)
     printf("Mes[%2d]: %s\n",i,nomeMes[i]);

  system("Pause");
}
```

CÓDIGO

```
(teclado:)
Mes[ 0]: Janeiro
Mes[ 1]: Fevereiro
Mes[ 2]: Marco
...
Mes[11]: Dezembro

Mes[ 0]: Janeiro
Mes[ 1]: Fevereiro
Mes[ 3]: Marco
Mes[ 4]: Maio
Mes[ 5]: Junho
Mes[ 6]: Julho
Mes[ 7]: Agosto
Mes[ 8]: Setembro
Mes[ 9]: Outubro
Mes[10]: Novembro
Mes[11]: Dezembro
```

SAÍDA

PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. entrada e saída de dados

Estudo de caso : (16C)

Desenhe um programa que leia um (texto de um) ficheiro e o apresente no ecrã.

strings. entrada e saída de dados

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
  FILE *fpIn;
  char buffer[101];

  fpIn = fopen("DocSurpresa.txt","r");
  if (fpIn == NULL) return;

  while(fgets(buffer,sizeof(buffer)-1,fpIn) != NULL)
    fputs(buffer,stdout);

  fclose(fpIn);
  system("Pause");
}
```

CÓDIGO

```
William Shakespeare (1564 - 1616)
William Shakespeare was an English
poet and playwright, often called
England's national poet and the
"Bard of Avon". He was born and
raised in Stratford-upon-Avon, a
small country town. He was John
Shakespeare's son, a successful
glover and alderman. His mother was
Mary Arden, a daughter of the
gentry. England's celebration of
their patron Saint George is on 23
April, which is also the day
claimed to be the birth date of
Shakespeare. The infant William was
baptised on 26 April 1564.
Few records of Shakespeare's
private life survive. Around the
age of eleven Shakespeare probably
entered the grammar school of
Stratford, King's New School, where
he studied theatre and acting, as
well as Latin literature and
history. When he finished school he
might have apprenticed for a time …
```

SAÍDA

funções para manipulação de caracteres & *strings*

14

strings. manipulação

Manipulação de *strings* e caracteres

A linguagem C tem um conjunto alargado de funções para manipulação de caracteres e *strings*.

Exemplos

strlen(str) : determina o comprimento de uma string
strcpy(dst,src) : copia uma *string* de src para dst
strcmp(str1,str2) : compara str1 e str2

Estas funções encontram-se nas bibliotecas de funções ctype.h e string.h

strings. manipulação

Biblioteca de manipulação de caracteres (em <ctype.h>)
Cada função tem como argumento um caracter ou EOF.

Protótipo	Descrição
`int isdigit( int c )`	Returns **true** if **c** is a digit and **false** otherwise.
`int isalpha( int c )`	Returns **true** if **c** is a letter and **false** otherwise.
`int isalnum( int c )`	Returns **true** if **c** is a digit or a letter and **false** otherwise.
`int isxdigit( int c )`	Returns **true** if **c** is a hexadecimal digit character and **false** otherwise.
`int islower( int c )`	Returns **true** if **c** is a lowercase letter and **false** otherwise.
`int isupper( int c )`	Returns **true** if **c** is an uppercase letter; **false** otherwise.
`int tolower( int c )`	If **c** is an uppercase letter, **tolower** returns **c** as a lowercase letter. Otherwise, **tolower** returns the argument unchanged.
`int toupper( int c )`	If **c** is a lowercase letter, **toupper** returns **c** as an uppercase letter. Otherwise, **toupper** returns the argument unchanged.
`int isspace( int c )`	Returns **true** if **c** is a white-space character—newline ('**\n**'), space (' '), form feed ('**\f**'), carriage return ('**\r**'), horizontal tab ('**\t**'), or vertical tab ('**\v**')—and **false** otherwise
`int iscntrl( int c )`	Returns **true** if **c** is a control character and **false** otherwise.
`int ispunct( int c )`	Returns **true** if **c** is a printing character other than a space, a digit, or a letter and **false** otherwise.
`int isprint( int c )`	Returns **true** value if **c** is a printing character including space (' ') and **false** otherwise.
`int isgraph( int c )`	Returns **true** if **c** is a printing character other than space (' ') and **false** otherwise.

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad

Pedro Melo-Pinto 2018

strings. manipulação

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
  char ch;
  scanf("%c", &ch);

  printf("\n%s%c%s%s\n", "De acordo com isdigit: ",ch,
    isdigit(ch) ? " e um " : " nao e um ", "digito");
  printf("%s%c%s%s\n", "De acordo com isalpha: ",ch,
    isalpha(ch) ? " e uma " : " nao e uma ", "letra");
  printf("%s%c%s%s\n", "De acordo com isalnum: ",ch,
    isalnum(ch) ? " e " : " nao e ", "digito ou letra");
  printf("%s%c%s%s\n", "De acordo com isxdigit: ",ch,
    isxdigit(ch) ? " e um " : " nao e um ", "digito hex");

  system("Pause");
}
```

CÓDIGO

```
(teclado:)
6
... isdigit: 6 e um digito
... isalpha: 6 nao e uma letra
... isalnum: 6 e digito ou letra
... isxdigit: 6 e um digito hex
--------------------------------
(teclado:)
F
... isdigit: F nao e um digito
... isalpha: F e uma letra
... isalnum: F e digito ou letra
... isxdigit: F e um digito hex
--------------------------------
(teclado:)
#
... isdigit: # nao e um digito
... isalpha: # nao e uma letra
... isalnum: # nao e digito ou letra
... isxdigit: # nao e um digito hex
--------------------------------
(teclado:)
s
... isdigit: s nao e um digito
... isalpha: s e uma letra
... isalnum: s e digito ou letra
... isxdigit: s nao e um digito hex
```

SAÍDA

PROGRAMAÇÃO DE COMPUTADORES

utad

Pedro Melo-Pinto 2018

strings. manipulação

Funções de conversão (em <stdlib.h>)
Converte *strings* em dígitos (valores numéricos – *integer* e *floating-point*).

Protótipo	Descrição
`double atof( const char *nPtr )`	Converts the string **nPtr** to **double**.
`int atoi( const char *nPtr )`	Converts the string **nPtr** to **int**.
`long atol( const char *nPtr )`	Converts the string **nPtr** to long **int**.
`double strtod( const char *nPtr, char **endPtr )`	Converts the string **nPtr** to **double**.
`long strtol( const char *nPtr, char **endPtr, int base )`	Converts the string **nPtr** to **long**.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base )`	Converts the string **nPtr** to **unsigned long**.
`double atof( const char *nPtr )`	Converts the string **nPtr** to **double**.
`int atoi( const char *nPtr )`	Converts the string **nPtr** to **int**.

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. manipulação

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
  char str[10];
  int strInt;
  float strFloat;

  printf("Introduza um numero: ");
  scanf("%s", str);

  strInt=atoi(str);
  strFloat=atof(str);

  printf("\n%s%s%s\n", "A string ",str, " convertida para:");
  printf("%s%d\n", "int dividida por 2 vale ",strInt/2);
  printf("%s%.2f\n", "float dividida por 2 vale ",strFloat/2);

  system("Pause");
}
```

CÓDIGO

```
(teclado:)
Introduza um numero: 56.7

A string 56.7 convertida para:
int dividida por 2 vale 28
float dividida por 2 vale 28.35
```

SAÍDA

PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. manipulação

Funções (em <stdio.h>)
Utilizadas para manipular caracteres e *strings*.

Protótipo	Descrição
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array **s** until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
`int putchar( int c );`	Prints the character stored in **c**.
`int puts( const char *s );`	Prints the string **s** followed by a newline character.
`int sprintf( char *s, const char *format,...);`	Equivalent to **printf**, except the output is stored in the array **s** instead of printing it on the screen.
`int sscanf( char *s, const char *format,...);`	Equivalent to **scanf**, except the input is read from the array **s** instead of reading it from the keyboard.
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array **s** until a newline or end-of-file character is encountered. A terminating null character is appended to the array.

strings. entrada e saída de dados

Estudo de caso : (16D)

(Utilização de *arrays* de *strings*)
Desenhe um programa que leia do teclado uma frase e a apresente ao contrário (do fim para o início) no ecrã.

strings. manipulação

```
#include <stdio.h>
#include <stdlib.h>

void printInvStr(char *sPtr);

main()
{
  char str[50];

  printf("Introduza uma frase:\n");
  gets(str);

  printf("\n%s\n", "A frase escrita ao contrario:");
  printInvStr(str);

  system("Pause");
}

void printInvStr(char *sPtr)
{
  if ( sPtr[0] == '\0' ) return;
  else
  {
      printInvStr( &sPtr[1] );
      putchar( sPtr[0] );
  }
}
```

CÓDIGO

```
(teclado:)
Introduza uma frase:
Um dia de sol.

A frase escrita ao contrario:
.los ed aid mU
```

SAÍDA

strings. manipulação

Funções de manipulação de *strings* (em <string.h>)
Manipulação dos dados.
Determinar o comprimento de uma *string*.

Protótipo	Descrição
`char *strcpy( char *s1, const char *s2 )`	Copies string **s2** into array **s1**. The value of **s1** is returned.
`char *strncpy(char *s1,const char *s2,size_t n )`	Copies at most **n** characters of string **s2** into array **s1**. The value of **s1** is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`char *strncat(char *s1,const char *s2, size_t n)`	Appends at most **n** characters of string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`size_t strlen( const char *s );`	Returns the number of characters (before NULL) in string s

strings. manipulação

Funções de manipulação de *strings* (em <string.h>)
Determinar o comprimento de uma *string*

int strlen(char *str)
* retorna o comprimento da *string* passada como argumento
* conta o número de caracteres até \0 (não conta este)

Exemplo:
char str = “Paulo”;
strlen(str) retorna o valor 5

strings. manipulação

Funções de manipulação de *strings* (em <string.h>)
Cópia de *strings*

char *strcpy(char *dst, char *src)
- copia os caracteres (incluindo \0) da *string* src para a *string* dst
- dst deve ter espaço suficiente (senão escreve em cima do que estiver após dst)
- se as duas strings se sobrepõem - parcial ou totalmente - (e.g., copiar uma string para ela mesma) o resultado é imprevisível
- o valor de retorno é a *string* de destino (dst)

char *strncpy(char *dst, char *src, int n)
- equivalente a strcpy, mas a cópia pára ao fim de n caracteres
- se forem copiados n não-null (não \0) caracteres, o caracter \0 não é copiado

strings. manipulação

CÓDIGO

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main()
{
  char str1[50], str2[50], str3[50];

  printf("Introduza uma frase:\n");
  gets(str1);
  strcpy(str3,str1);
  printf("Introduza outra frase:\n");
  gets(str2);

  printf("\n%s\n%s","As duas frases concatenadas:",
    strcat(str1,str2));
  printf("\nAs 2 frases concatenadas com %1d char da 2:\n%s",
    7, strncat(str3,str2,7));

  system("Pause");
}
```

SAÍDA

```
(teclado:)
Introduza uma frase:
Um dia de sol
Introduza outra frase:
 brilhante.

As duas frases concatenadas:
Um dia de sol brilhante.
As 2 frases concatenadas com 7 char da 2:
Um dia de sol brilha
```

strings. entrada e saída de dados

Estudo de caso : (16D) revisitado

(Utilização de *arrays* de *strings*)
Desenhe um programa que leia do teclado uma frase e a apresente ao contrário (do fim para o início) no ecrã.

strings. manipulação

```
…
#include <string.h>

char * invStr(char * sPtr);

main()
{
  char str[50];

  printf("Introduza uma frase:\n"); gets(str);

  printf("\n%s\n%s","A frase escrita ao contrario vale:",invStr(str));

  system("Pause");
}

char * invStr(char * sPtr)
{
  char str[50];

  if ( sPtr[0] == '\0' ) return("\0");
  for(int i=0; i<strlen(sPtr); i++)
      str[strlen(sPtr)-i-1]=sPtr[i];
  str[i]='\0';

  return(str);
}
```

CÓDIGO

```
(teclado:)
Introduza uma frase:
Um dia de sol.

? str de invStr() é uma variável local.
```

SAÍDA

exemplo PROGRAMAÇÃO DE COMPUTADORES

strings. manipulação

```
…
#include <string.h>

void invStr(char * inStr, char *outStr);

main()
{
  char inS[50], outS[50];

  printf("Introduza uma frase:\n"); gets(inS);

  invStr(inS, outS);
  printf("\n%s\n%s", "A frase escrita ao contrario vale:",outS);

  system("Pause");
}

void invStr(char * inStr, char *outStr)
{
  if ( inStr[0] == '\0' ) {outStr[0]='\0'; return; }

  for(int i=0; i<strlen(inStr); i++)
     outStr[strlen(inStr)-i-1]=inStr[i];
  outStr[strlen(inStr)]='\0';

  return;
}
```

CÓDIGO

```
(teclado:)
Introduza uma frase:
Um dia de sol.

A frase escrita ao contrario:
.los ed aid mU
```

SAÍDA

PROGRAMAÇÃO DE COMPUTADORES

strings. manipulação

Funções de comparação de *strings* (em <string.h>)
Compara códigos numéricos (ASCII) da *string*.

Protótipo	Descrição
`int strcmp(const char *s1, const char *s2)`	Compares string **s1** to **s2**. Returns a negative number (**s1 < s2**), zero (**s1 == s2**), or a positive number (**s1 > s2**)
`int strncmp(const char *s1, const char *s2, size_t n)`	Compares up to **n** characters of string **s1** to **s2**. Returns values as above

Funções de pesquisa em *strings* (em <string.h>)

Protótipo	Descrição
`char *strchr(const char *s, int c)`	Locates the first occurrence of character **c** in string **s**. If **c** is found, a pointer to **c** in **s** is returned. Otherwise, a **NULL** pointer is returned.
`size_t strcspn(const char *s1, const char *s2)`	Determines and returns the length of the initial segment of string **s1** consisting of characters not contained in string **s2**.
`size_t strspn(const char *s1, const char *s2)`	Determines and returns the length of the initial segment of string **s1** consisting only of characters contained in string **s2**.
`char *strpbrk(const char *s1, const char *s2)`	Locates the first occurrence in string **s1** of any character in string **s2**. If a character from string **s2** is found, a pointer to the character in string **s1** is returned. Otherwise, a **NULL** pointer is returned.
`char *strrchr(const char *s, int c)`	Locates the last occurrence of **c** in string **s**. If **c** is found, a pointer to **c** in string **s** is returned. Otherwise, a **NULL** pointer is returned.
`char *strstr(const char *s1,const char *s2)`	Locates the first occurrence in string **s1** of string **s2**. If the string is found, a pointer to the string in **s1** is returned. Otherwise, a **NULL** pointer is returned.
`char *strtok(char *s1, const char *s2)`	A sequence of calls to **strtok** breaks string **s1** into "tokens" - logical pieces such as words in a line of text - separated by characters contained in string **s2**. The first call contains **s1** as the first argument, and subsequent calls to continue tokenizing the same string contain **NULL** as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, **NULL** is returned.

strings. manipulação

Funções de manipulação de *strings* (em <string.h>)
Comparar *strings*

int strcmp(char *str1, char *str2)
* retorna < 0
  se o valor ASCII do caracter em que diferem (sequencialmente) é menor em str1
  ou se str1 é igual (sequencialmente) a str2 (mas str2 é mais comprida)
* retorna > 0
  se o valor ASCII do caracter em que diferem (sequencialmente) é maior em str1
  ou se str1 é igual (sequencialmente) a str2 (mas str1 é mais comprida)
* retorna 0
  se str1 e str2 não diferem

Exemplos:
strcmp("hello","hello") - retorna 0
strcmp("yello","hello") - retorna um valor > 0
strcmp("Hello","hello") - retorna um valor < 0
strcmp("hello","hello there") - retorna um valor < 0
strcmp("some diff","some dift") - retorna um valor < 0

strings. manipulação

Funções de manipulação de *strings* (em <string.h>)
Comparar *strings*

int strncmp(char *str1, char *str2, int n)
* equivalente a strcmp, mas a comparação é feita para os primeiros n caracteres
* se uma das strings for mais curta do que n caracteres utiliza esse comprimento

Exemplos:
strcmp("some diff","some DIFF") - retorna um valor > 0
strncmp("some diff","some DIFF",4) - retorna 0

int strcasecmp(char *str1, char *str2)
* equivalente a strcmp, mas em que um caracter em minúscula ou maiúscula (e.g., 'a' e 'A') é considerado igual

int strncasecmp(char *str1, char *str2, int n)
* versão de strncmp, em que um caracter em minúscula ou maiúscula é considerado igual

strings. manipulação

Funções de manipulação de *strings* (em <string.h>)
Concatenar (reunir) *strings*

char *strcat(char *dstS, char *addS)
- acrescenta addS à *string* dstS (no final desta)
- retorna dstS
- pode ser problemático se a string resultante for demasiado longa para dstS

char *strncat(char *dstS, char *addS, int n)
- acrescenta os primeiros n caracteres de addS à *string* dstS (no final desta)
- se addS tiver menos do que n caracteres só os caracteres de addS são acrescentados
- acrescenta sempre \0

strings. manipulação

Funções de manipulação de *strings* (em <string.h>)
Procurar em *strings*

char *strchr(char *str, int ch)
- retorna um apontador (char *) para a primeira ocorrência de ch na string str
- retorna NULL se ch não existe em str
- Podemos usar aritmética de apontadores (subtrair o apontador original do resultado) para determinar posição no array

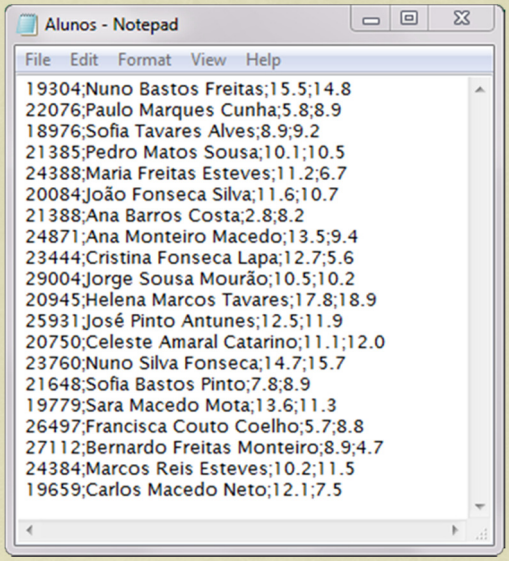
char *strstr(char *str, char *searchstr)
- Equivalente a strchr mas procura a primeira ocorrência da string searchstr em str

char *strrchr(char *str, int ch)
- Equivalente a strchr mas procura a primeira ocorrência de ch do fim para o início de str

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. manipulação



Seja o ficheiro “Alunos.txt”

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. manipulação

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void main()
{
  FILE *fpIn;
  char buffer[1000], noCh[1000];
  char *pos, ch;

  fpIn = fopen(“Alunos.txt","r"); if (fpIn == NULL) return;

  printf("Introduza um char: "); scanf("%c", &ch);

  while (fgets(buffer,sizeof(buffer)-1,fpIn) != NULL) {
    if ((pos = strchr(buffer,ch)) != NULL)
    {
      strncpy(noCh,buffer,(pos - buffer));
      noCh[pos - buffer] = '\0';
      printf("%s\n",noCh);
    }
    else
      printf("%s",buffer);
  }

  fclose(fpIn);
  system("PAUSE");
}
```

CÓDIGO

```
(teclado:)
Introduza um char: ;

19304
22076
18976
21385
24388
20084
21388
24871
23444
29004
20945
...
```

SAÍDA

strings. manipulação

```
…
void main()
{
  FILE *fpIn;
  char buffer[1000]; char noCh[1000];
  char *pos1, *pos2, ch;

  fpIn = fopen("Alunos.txt","r"); if (fpIn == NULL) return;

  printf("Introduza um char: "); scanf("%c", &ch);

  while (fgets(buffer,sizeof(buffer)-1,fpIn) != NULL) {
    if ((pos1 = strchr(buffer,ch)) != NULL) {
       if ((pos2 = strchr(pos1+1,ch)) != NULL) {
         strncpy(noCh,pos1+1,(pos2 - pos1 - 1));
         noCh[pos2 - pos1 - 1] = '\0';
       }
       else {
         strncpy(noCh,pos1+1,(pos1 - buffer));
         noCh[pos1 - buffer] = '\0';
       }
       printf("%s\n",noCh);
    }
    else printf("%s",buffer);
  }

  fclose(fpIn); system("PAUSE");
}
```

CÓDIGO

```
(teclado:)
Introduza um char: ;

Nuno Bastos Freitas
Paulo Marques Cunha
Sofia Tavares Alves
Pedro Matos Sousa
Maria Freitas Esteves
João Fonseca Silva
Ana Barros Costa
Ana Monteiro Macedo
Cristina Fonseca Lapa
Jorge Sousa Mourão
Helena Marcos Tavares
…
```

SAÍDA

strings. manipulação, *parsing*

Estudo de caso : (17)

(*Parsing* de *strings*)
Desenhe um programa que leia do teclado uma frase e apresente no ecrã o array das palavras contidas na frase.

Nota: admita que o separador das palavras é o espaço.

strings. manipulação, *parsing*

Funções de manipulação de *strings* (em <string.h>)
Parsing de *strings*

Parsing
Dividir uma string em elementos (unidades) baseado em delimitadores conhecidos
Um elemento/símbolo é a unidade de informação ("palavra")
Os delimitadores são um (ou mais) caracteres utilizados para separar os elementos

char *strtok(char *str, char *delimiters)
* se a função for chamada com uma *string* como primeiro argumento, encontra a primeira ocorrência de uma *string* separada pelos delimitadores
* se a função for chamada com NULL como primeiro argumento, encontra a próxima ocorrência de uma *string* separada pelos delimitadores
* a função strtok() efectua alterações na string original, pelo que é melhor fazer uma cópia se necessário

strings. manipulação, *parsing*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main()
{
  int i=0, n=0;
  char str[50], *aTokens[10];

  printf("Introduza uma frase:\n");
  gets(str);

  aTokens[n] = strtok(str," ");
  while(aTokens[n] != NULL)
    aTokens[++n] = strtok(NULL," ");

  for(i=0; i<n; i++)
      printf("\nelemento %2d: %s", i, aTokens[i]);

  system("Pause");
}
```

CÓDIGO

```
(teclado:)
Introduza uma frase:
Uma frase assim nao se inventa

elemento 0: Uma
elemento 1: frase
elemento 2: assim
elemento 3: nao
elemento 4: se
elemento 5: inventa
```

SAÍDA

strings. manipulação, *parsing*

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main()
{
  int i=0, n=0;
  char str[50], *aTokens[10];

  printf("Introduza uma frase:\n");
  gets(str);

  aTokens[n] = strtok(str," ");
  while(aTokens[n] != NULL)
    aTokens[++n] = strtok(NULL," ");

  for(i=0; i<n; i++)
      printf("\nelemento %2d: %s", i, aTokens[i]);

  system("Pause");
}
```

CÓDIGO

```
(teclado:)
Introduza uma frase:
Uma frase assim nao se inventa

elemento 0: Uma
elemento 1: frase
elemento 2: assim
elemento 3: nao
elemento 4: se
elemento 5: inventa
----------------------------
(teclado:)
Introduza uma frase:
( 3 + 8 ) * 2

elemento 0: (
elemento 1: 3
elemento 2: +
elemento 3: 4
elemento 4: )
elemento 5: *
elemento 6: 2
```

SAÍDA

strings. manipulação, *parsing*

Estudo de caso : (17B)

(*Parsing* de *strings*)
Desenhe um programa que leia do teclado uma expressão algébrica simples e apresente o resultado no ecrã.

Notas
Admita que os operandos são números inteiros de um só dígito.
Considere os seguintes operadores/delimitadores (binários):
`+ - * / ^ ( )`

strings. manipulação, *parsing*

Estudo de caso : (17B)

(*Parsing* de *strings*)
Desenhe um programa que leia do teclado uma expressão algébrica simples e apresente o resultado no ecrã.

Parte I : *parsing* e conversão para *postfix*

Notas
Admita que os operandos são números inteiros de um só dígito.
Considere os seguintes operadores/delimitadores (binários):
`+ - * / ^ ( )`

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Uma expressão algébrica é uma combinação de operadores e operandos.
Um operador é o símbolo matemático ou lógico que representa a operação a ser feita.
Exemplos de operadores:
+ , - , * , / , ^
Um operando é a entidade/quantidade sobre a qual a operação vai ser realizada.
Um operando pode ser uma variável (e.g. x, y, z etc.) ou uma constante (e.g. 4, 0, 9.1, etc.).

Exemplo de uma expressão:
x + y * z

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Notações

INFIX: Os operandos ladeiam o operador.
e.g. x + y * 3

POSTFIX: Também conhecida como Reverse Polish Notation (RPN).
O operador vem depois dos operandos.
e.g. x y + (equivalente a x + y) ou x y z + * (equivalente a x * (y + z))

PREFIX: Também conhecida como Polish Notation (PN).
O operador vem antes dos operandos.
e.g. + x y (equivalente a x + y) ou * + x y z (equivalente a (x + y) * z)

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Notações

Porquê utilizar estas notações (PREFIX e POSTFIX) quando temos as expressões habituais (INFIX) ?

As expressões INFIX são difíceis de calcular (com recurso a uma *string* de entrada), pois para tal precisamos de ter em conta a prioridade dos operadores e a propriedade associativa.

exemplo : a expressão 3+5*4 pode ter como resultado 32 se considerarmos (3+5)*4
ou 23 se considerarmos 3+(5*4).

Eis porque se torna necessário definir prioridades para os operadores.
A precedência dos operadores dita a ordem de cálculo. Operadores com maior prioridade são aplicados antes dos de menor prioridade.

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas
Prioridades

Como determinar os operandos de um determinado operador ?
a + b * c ou a * b + c / d

Faz-se atribuindo prioridades aos operadores.
prioridade(*) = prioridade(/) > prioridade(+) = prioridade(-)

- Quando um operando fica entre dois operadores é associado àquele que tem prioridade maior.
- Quando um operando fica entre dois operadores de prioridade idêntica, é associado ao que está à sua esquerda.
- Subexpressões entre delimitadores (parênteses) são tratadas como um operando simples, independentemente da restante expressão.

  a * b / (c + d) → c + d é calculado antes de tudo o resto

strings. manipulação, *parsing*

Infix notation is easy to read for *humans*, whereas pre-/postfix notation is easier to parse for a machine. The big advantage in pre-/postfix notation is that there never arise any questions like operator precedence.

Parsing de *strings*. Expressões algébricas
Infix vs Postfix

Como vimos as operações INFIX são mais difíceis de calcular
- Precisam de prioridades dos operadores
- Precisam de regras em caso de igual prioridade
- Precisam de delimitadores (parênteses – propriedade associativa) caso necessitemos de alterar a ordem de cálculo

As expressões em notação PREFIX ou POSTFIX não apresentam estas necessidades

exemplos

INFIX	POSTFIX
a + b	a b +
a / b * c	a b / c *
a * (b + c)	a b c + *
(a + b) * (c + d)	a b + c d + *
a – b / (c * d ^ e)	a b c d e ^ * / -

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas
Conversão Infix para Postfix

01. Entrada de dados - Expressão em notação Infix
(introduzida pelo utilizador através do teclado, por exemplo).
02. Converter para notação Postfix
03. Calcular o valor da expressão sem recurso a prioridades ou parênteses
Estas expressões são calculadas da esquerda para a direita

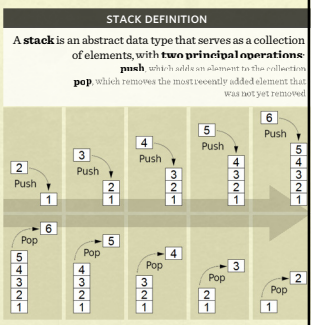
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas
Algoritmo de conversão Infix para Postfix

Entrada de dados: expressão em notação Infix

1. Examinar o próximo elemento da string de entrada.
2. Se operando enviar para a string de saída.
3. Se parêntese de abertura enviar (push) para a stack.
4. Se operador então
   I. Se stack vazia enviar operador para a stack
   II. Se top stack é parêntese de abertura enviar operador para a stack
   III. Se tem maior prioridade do que top stack enviar operador para a stack
   IV. Se não for nenhum dos anteriores retirar o operador da stack e enviar para string de saída; repetir 4
5. Se parêntese de fecho retirar os operadores da stack enviar para string de saída até encontrar um parêntese de abertura; retirar da stack (e ignorar) este parêntese.
6. Se string de entrada ainda tem elementos ir para 1.
7. Se string de entrada não tem mais elementos, retirar da stack (pop) para a *string* de saída os restantes operadores.

strings. manipulação, *parsing*



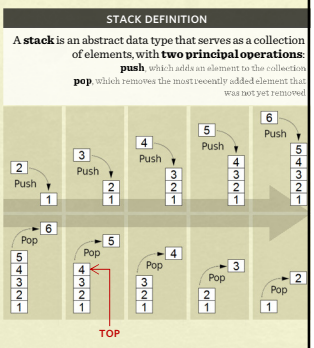
Parsing de *strings*. Expressões algébricas

Algoritmo de conversão Infix para Postfix

Entrada de dados: expressão em notação Infix

1. Examinar o próximo elemento da string de entrada.
2. Se operando enviar para a string de saída.
3. Se parêntese de abertura enviar (push) para a stack.
4. Se operador então
   I. Se stack vazia enviar operador para a stack
   II. Se top stack é parêntese de abertura enviar operador para a stack
   III. Se tem maior prioridade do que top stack enviar operador para a stack
   IV. Se não for nenhum dos anteriores retirar o operador da stack e enviar para string de saída; repetir 4
5. Se parêntese de fecho retirar os operadores da stack enviar para string de saída até encontrar um parêntese de abertura; retirar da stack (e ignorar) este parêntese.
6. Se string de entrada ainda tem elementos ir para 1.
7. Se string de entrada não tem mais elementos, retirar da stack (pop) para a *string* de saída os restantes operadores.

strings. manipulação, *parsing*



Parsing de *strings*. Expressões algébricas

Algoritmo de conversão Infix para Postfix

Entrada de dados: expressão em notação Infix

1. Examinar o próximo elemento da string de entrada.
2. Se operando enviar para a string de saída.
3. Se parêntese de abertura enviar (push) para a stack.
4. Se operador então
   I. Se stack vazia enviar operador para a stack
   II. Se top stack é parêntese de abertura enviar operador para a stack
   III. Se tem maior prioridade do que top stack enviar operador para a stack
   IV. Se não for nenhum dos anteriores retirar o operador da stack e enviar para string de saída; repetir 4
5. Se parêntese de fecho retirar os operadores da stack enviar para string de saída até encontrar um parêntese de abertura; retirar da stack (e ignorar) este parêntese.
6. Se string de entrada ainda tem elementos ir para 1.
7. Se string de entrada não tem mais elementos, retirar da stack (pop) para a *string* de saída os restantes operadores.

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Exemplo de conversão Infix para Postfix

converter a expressão 2*3/(2-1)+5*3 para notação Postfix

Expressão em notação Infix

2 * 3 / (2 - 1) + 5 * 3 →

Expressão	Stack	Saída	
2	Vazia	2	2
*	*	2	4 I.
3	*	23	2
/	/	23*	4 IV. 4 I.
(	/(	23*	3
2	/(	23*2	2
-	/(-	23*2	4 II.
1	/(-	23*21	2
)	/	23*21-	5
+	+	23*21-/	4 IV. 4 I.
5	+	23*21-/5	2
*	+*	23*21-/53	4 III.
3	+*	23*21-/53	2
Vazia	Vazia	23*21-/53*+	7

1. Examinar o próximo elemento da *string* de entrada.
2. Se operando enviar para a *string* de saída.
3. Se parêntese de abertura enviar (*push*) para a *stack*.
4. Se operador então
   I. Se *stack* vazia enviar operador para a *stack*
   II. Se *top stack* é parêntese de abertura enviar operador para a *stack*
   III. Se tem maior prioridade do que *top stack* enviar operador para a *stack*
   IV. Se não for nenhum dos anteriores retirar o operador da *stack* e enviar para *string* de saída; repetir 4
5. Se parêntese de fecho retirar os operadores da *stack* enviar para *string* de saída até encontrar um parêntese de abertura; retirar da *stack* (e ignorar) este parêntese.
6. Se *string* de entrada ainda tem elementos ir para 1.
7. Se *string* de entrada não tem mais elementos, retirar da *stack* (*pop*) para a *string* de saída os restantes operadores.

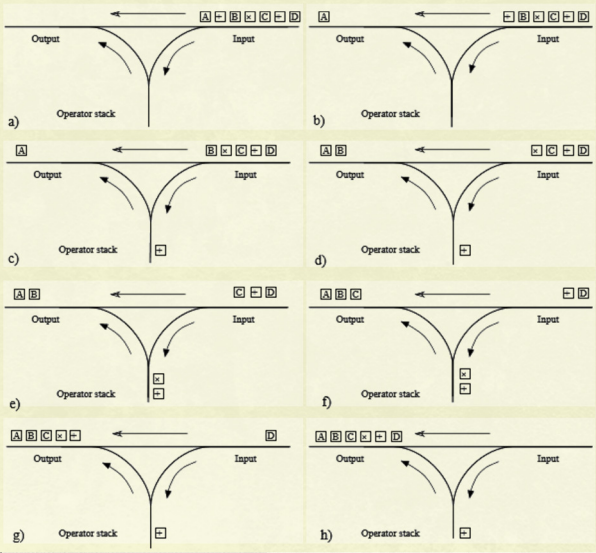
Expressão em notação Postfix

→ **2 3 * 2 1 - / 5 3 * +**

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Exemplo de conversão Infix para Postfix



Exemplo:

A + B * C + D

1. Examinar o próximo elemento da *string* de entrada.
2. Se operando enviar para a *string* de saída.
3. Se parêntese de abertura enviar (*push*) para a *stack*.
4. Se operador então
   I. Se *stack* vazia enviar operador para a *stack*
   II. Se *top stack* é parêntese de abertura enviar operador para a *stack*
   III. Se tem maior prioridade do que *top stack* enviar operador para a *stack*
   IV. Se não for nenhum dos anteriores retirar o operador da *stack* e enviar para *string* de saída; repetir 4
5. Se parêntese de fecho retirar os operadores da *stack* enviar para *string* de saída até encontrar um parêntese de abertura; retirar da *stack* (e ignorar) este parêntese.
6. Se *string* de entrada ainda tem elementos ir para 1.
7. Se *string* de entrada não tem mais elementos, retirar da *stack* (*pop*) para a *string* de saída os restantes operadores.

strings. manipulação, *parsing*

```
...

#define max_STACK 50
#define max_STR 50

int priority(char op);
void push(char stack[max_STACK],int *stackTop,char ch);
void pop(char stack[max_STACK],int *stackTop,char *ch);
void pushF(float stack[max_STACK],int *stackTop,float f);
void popF(float stack[max_STACK],int *stackTop,float *f);
float calculate(char outStr[max_STR]);

void main()
{
   char inStr[max_STR], outStr[max_STR], stack[max_STACK], ch;
   int stackTop = -1, i, j=0;

   printf("Expressao (ints de 1 digito):\n"); gets(inStr);

   for(i=0; i<strlen(inStr); i++)
   {
      switch(inStr[i])
      {
         case '(' :   push(stack,&stackTop,inStr[i]);
                      break;
         case ')' :   while(stack[stackTop] != '(') {
                         pop(stack,&stackTop,&ch);
                         outStr[j++]=ch;
                      }
                      pop(stack,&stackTop,&ch);
                      break;
         case ' ' :   break;
```

CÓDIGO

```
(teclado:)
Expressao (ints de 1 digito):
```

SAÍDA

strings. manipulação, *parsing*

```
...

#define max_STACK 50
#define max_STR 50

int priority(char op);
void push(char stack[max_STACK],int *stackTop,char ch);
void pop(char stack[max_STACK],int *stackTop,char *ch);
void pushF(float stack[max_STACK],int *stackTop,float f);
void popF(float stack[max_STACK],int *stackTop,float *f);
float calculate(char outStr[max_STR]);

void main()
{
   char inStr[max_STR], outStr[max_STR], stack[max_STACK], ch;
   int stackTop = -1, i, j=0;

   printf("Expressao (ints de 1 digito):\n"); gets(inStr);

   for(i=0; i<strlen(inStr); i++)
   {
      switch(inStr[i])
      {
         case '(' :   push(stack,&stackTop,inStr[i]);
                      break;
         case ')' :   while(stack[stackTop] != '(') {
                         pop(stack,&stackTop,&ch);
                         outStr[j++]=ch;
                      }
                      pop(stack,&stackTop,&ch);
                      break;
         case ' ' :   break;
```

CÓDIGO

```
(teclado:)
Expressao (ints de 1 digito):
```

Parsing de *strings* expressões algébricas

Algoritmo de conversão Infix para Postfix

1. Examinar o próximo elemento da *string* de entrada.
2. Se operando enviar para a *string* de saída.
3. Se parêntese de abertura enviar (*push*) para a *stack*.
4. Se operador então
   - I. Se *stack* vazia enviar operador para a *stack*
   - II. Se *top stack* é parêntese de abertura enviar operador para a *stack*
   - III. Se tem maior prioridade do que *top stack* enviar operador para a *stack*
   - IV. Se não for nenhum dos anteriores retirar o operador da *stack* e enviar para *string* de saída; repetir 4
5. Se parêntese de fecho retirar os operadores da *stack* enviar para *string* de saída até encontrar um parêntese de abertura; retirar da *stack* (e ignorar) este parêntese.
6. Se *string* de entrada ainda tem elementos ir para 1.
7. Se *string* de entrada não tem mais elementos, retirar da *stack* (*pop*) para a *string* de saída os restantes operadores.

SAÍDA

strings. manipulação, *parsing*

```
...
       case '^' :
       case '*' :
       case '/' :
       case '+' :
       case '-' :   if(stackTop == -1) push(stack,&stackTop,inStr[i]);
                    else if(priority(inStr[i]) > priority(stack[stackTop]))
                           push(stack,&stackTop,inStr[i]);
                         else {
                            while(priority(inStr[i]) <= priority(stack[stackTop])) {
                               if (stackTop == -1) break;
                               pop(stack,&stackTop,&ch);
                               outStr[j++]=ch;
                            }
                            push(stack,&stackTop,inStr[i]);
                         }
                    break;
       default  :   outStr[j++]=inStr[i];
    }
  }

  while(stackTop != -1) {
     pop(stack,&stackTop,&ch);
     outStr[j++]=ch;
  }
  outStr[j]='\0';

  printf("\nExpressao (Postfix):\n%s\n",outStr);
  //printf("\nResultado:\n%.2f\n\n",calculate(outStr));
  system("Pause");
}
```

CÓDIGO

```
(teclado:)
Expressao (ints de 1 digito):
```

SAÍDA

strings. manipulação, *parsing*

```
...
       case '^' :
       case '*' :
       case '/' :
       case '+' :
       case '-' :   if(stackTop == -1) push(stack,&stackTop,inStr[i]);
                    else if(priority(inStr[i]) > priority(stack[stackTop]))
                           push(stack,&stackTop,inStr[i]);
                         else {
                            while(priority(inStr[i]) <= priority(stack[stackTop])) {
                               if (stackTop == -1) break;
                               pop(stack,&stackTop,&ch);
                               outStr[j++]=ch;
                            }
                            push(stack,&stackTop,inStr[i]);
                         }
                    break;
       default  :   outStr[j++]=inStr[i];
    }
  }

  while(stackTop != -1) {
     pop(stack,&stackTop,&ch);
     outStr[j++]=ch;
  }
  outStr[j]='\0';

  printf("\nExpressao (Postfix):\n%s\n",outStr);
  //printf("\nResultado:\n%.2f\n\n",calculate(outStr));
  system("Pause");
}
```

CÓDIGO

```
(teclado:)
Expressao (ints de 1 digito):
```

Parsing de *strings* expressões algébricas
Algoritmo de conversão Infix para Postfix

1. Examinar o próximo elemento da *string* de entrada.
2. Se operando enviar para a *string* de saída.
3. Se parêntese de abertura enviar (*push*) para a *stack*.
4. Se operador então
   I. Se *stack* vazia enviar operador para a *stack*
   II. Se *top stack* é parêntese de abertura enviar operador para a *stack*
   III. Se tem maior prioridade do que *top stack* enviar operador para a *stack*
   IV. Se não for nenhum dos anteriores retirar o operador da *stack* e enviar para *string* de saída; repetir 4
5. Se parêntese de fecho retirar os operadores da *stack* enviar para *string* de saída até encontrar um parêntese de abertura; retirar da *stack* (e ignorar) este parêntese.
6. Se *string* de entrada ainda tem elementos ir para 1.
7. Se *string* de entrada não tem mais elementos, retirar da *stack* (*pop*) para a *string* de saída os restantes operadores.

SAÍDA

exemplo PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. manipulação, *parsing*

FUNÇÕES

```
...
int priority(char op)
{
  switch(op)
  {
    case '^' : return(4);
    case '*' :
    case '/' : return(3);
    case '+' :
    case '-' : return(2);
    case '(' : return(1);
    default  : return(0);
  }
}

void push(char stack[max_STACK],int *stackTop,char ch)
{
  if(*stackTop == max_STACK - 1) return;
  else { (*stackTop)++; stack[*stackTop]=ch; }
}

void pop(char stack[max_STACK],int *stackTop,char *ch)
{
  if (*stackTop == -1) return;
  else { *ch=stack[*stackTop]; (*stackTop)--; }
}
```

CÓDIGO

```
(teclado:)
Expressao (ints de 1 digito):
3 + 2

Expressao (Postfix):
32+
------------------------------
(teclado:)
Expressao (ints de 1 digito):
5^2 *( 3 +2)

Expressao (Postfix):
52^32+*
```

SAÍDA

PROGRAMAÇÃO DE COMPUTADORES

utad
Pedro Melo-Pinto 2018

strings. manipulação, *parsing*

Estudo de caso : (17B)

(*Parsing* de *strings*)
Desenhe um programa que leia do teclado uma expressão algébrica simples e apresente o resultado no ecrã.

Parte II : cálculo

Notas
Admita que os operandos são números inteiros de um só dígito.
Considere os seguintes operadores/delimitadores (binários):
`+ - * / ^ ( )`

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)

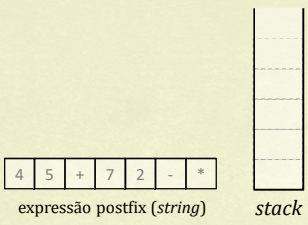
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



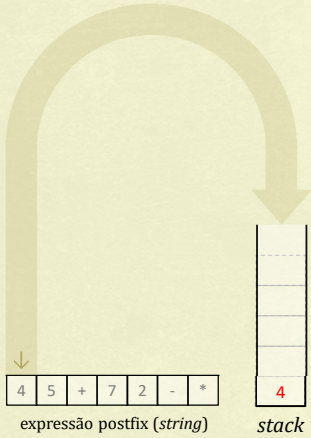
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



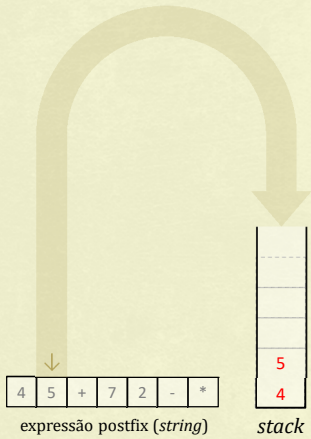
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)

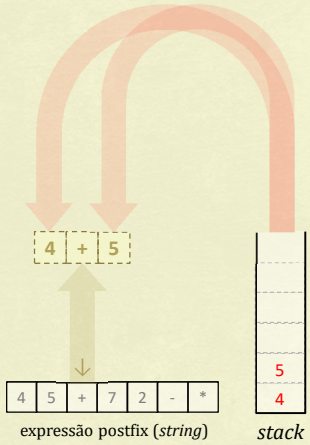


strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas
Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)

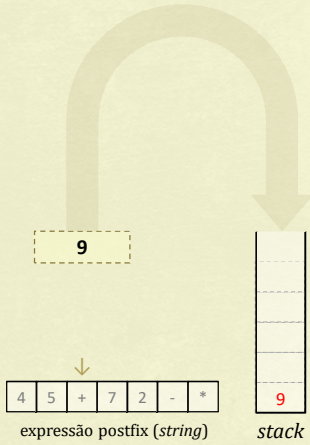


strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas
Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



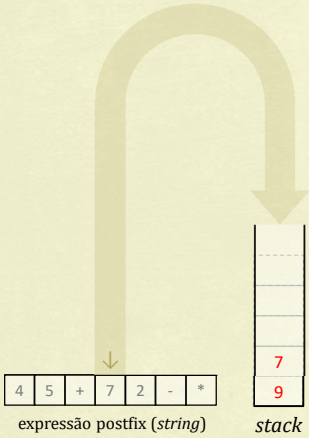
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



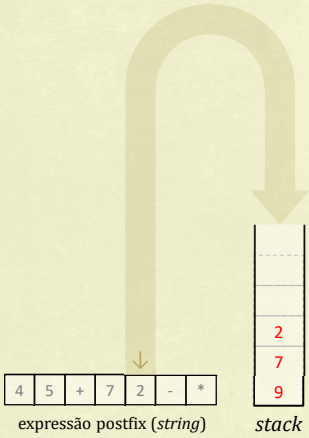
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



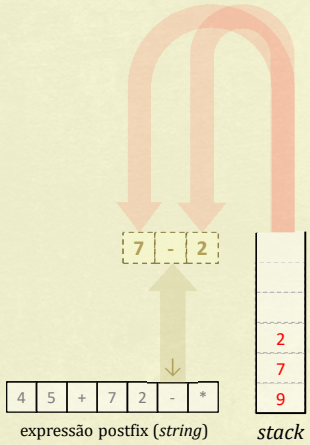
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



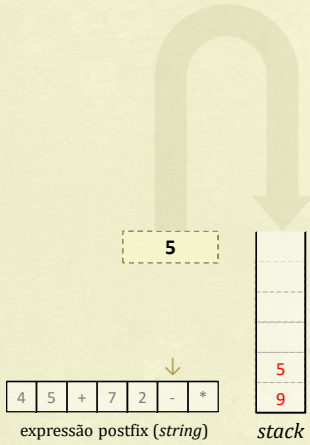
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



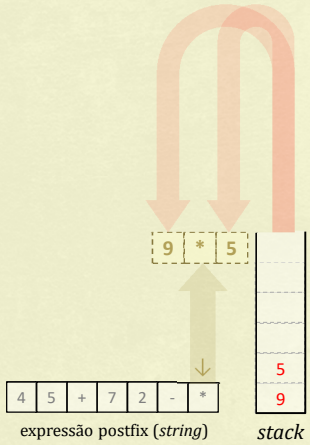
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



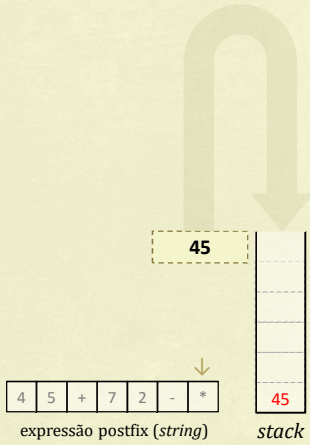
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



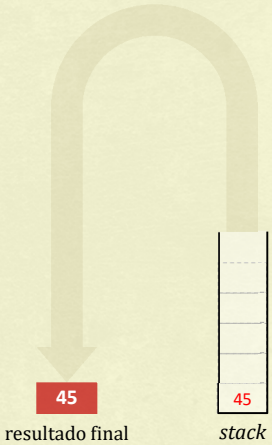
strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Cálculo de expressão Postfix

- Uma expressão POSTFIX calcula-se percorrendo a *string* da esquerda para a direita.
- Cada operador numa expressão POSTFIX aplica-se aos dois operandos imediatamente anteriores (na *string*).

- Vamos utilizar uma *stack* para este cálculo.
- Cada vez que for lido um operando (variável ou constante), introduzimo-lo na *stack* (*push*).
- Cada vez que for lido um operador, os respectivos operandos serão os dois elementos no topo da *stack*, que teremos de substituir pelo resultado da operação.
  (*pop* 2x da stack → realizar operação → *push* resultado)



resultado final *stack*

strings. manipulação, *parsing*

Parsing de *strings*. Expressões algébricas

Algoritmo de cálculo de expressão Postfix

Entrada de dados: expressão em notação Postfix

1. Considerar uma stack vazia.
2. Ler próximo elemento da string de entrada.
   I. Se é número enviar para a stack
   II. Senão (é operador) retirar dois elementos da stack, aplicar o operador e enviar resultado para a stack
3. Se string de entrada ainda tem elementos ir para 2.
4. Se string de entrada não tem mais elementos, retirar da stack (pop) para a *string* de saída a resposta.

strings. manipulação, *parsing*

+ FUNÇÕES

```
...
float calculate(char outStr[max_STR])
{
  int i, stackTop = -1;
  float stack[max_STACK],a,b;
  char str[2];

  for(i=0; i<strlen(outStr); i++)
    switch(outStr[i])
    {
      case '+': popF(stack,&stackTop,&b);popF(stack,&stackTop,&a);
                pushF(stack,&stackTop,a+b);break;
      case '-': popF(stack,&stackTop,&b);popF(stack,&stackTop,&a);
                pushF(stack,&stackTop,a-b);break;
      case '*': popF(stack,&stackTop,&b);popF(stack,&stackTop,&a);
                pushF(stack,&stackTop,a*b);break;
      case '/': popF(stack,&stackTop,&b);popF(stack,&stackTop,&a);
                pushF(stack,&stackTop,a/b);break;
      case '^': popF(stack,&stackTop,&b);popF(stack,&stackTop,&a);
                pushF(stack,&stackTop,pow(a,b));break;
      default:  str[0]=outStr[i];str[1]='\0';
                pushF(stack,&stackTop,atof(str));break;
    }
  popF(stack,&stackTop,&a);
  return(a);
}
```

CÓDIGO

```
(teclado:)
Expressao (ints de 1 digito):
```

SAÍDA

strings. manipulação, *parsing*

```
...

#define max_STACK 50
#define max_STR 50

int priority(char op);
void push(char stack[max_STACK],int *stackTop,char ch);
void pop(char stack[max_STACK],int *stackTop,char *ch);
void pushF(float stack[max_STACK],int *stackTop,float f);
void popF(float stack[max_STACK],int *stackTop,float *f);
float calculate(char outStr[max_STR]);

void main()
{
  char inStr[max_STR], outStr[max_STR], stack[max_STACK], ch;
  int stackTop = -1, i, j=0;

  printf("Expressao (ints de 1 digito):\n"); gets(inStr);

  for(i=0; i<strlen(inStr); i++)
  {
    switch(inStr[i])
    {
      case '(' :  push(stack,&stackTop,inStr[i]);
                  break;
      case ')' :  while(stack[stackTop] != '(') {
                    pop(stack,&stackTop,&ch);
                    outStr[j++]=ch;
                  }
                  pop(stack,&stackTop,&ch);
                  break;
      case ' ' :  break;
```

CÓDIGO

```
(teclado:)
Expressao (ints de 1 digito):
```

SAÍDA

strings. manipulação, *parsing*

```
...
       case '^' :
       case '*' :
       case '/' :
       case '+' :
       case '-' :   if(stackTop == -1) push(stack,&stackTop,inStr[i]);
                    else if(priority(inStr[i]) > priority(stack[stackTop]))
                           push(stack,&stackTop,inStr[i]);
                         else {
                           while(priority(inStr[i]) <= priority(stack[stackTop])) {
                             if (stackTop == -1) break;
                             pop(stack,&stackTop,&ch);
                             outStr[j++]=ch;
                           }
                           push(stack,&stackTop,inStr[i]);
                         }
                    break;
       default  :   outStr[j++]=inStr[i];
     }
   }

   while(stackTop != -1) {
     pop(stack,&stackTop,&ch);
     outStr[j++]=ch;
   }
   outStr[j]='\0';

   printf("\nExpressao (Postfix):\n%s\n",outStr);
   printf("\nResultado: %.2f\n\n",calculate(outStr));
   system("Pause");
}
```

CÓDIGO

```
(teclado:)
Expressao (ints de 1 digito):
3 + 2

Expressao (Postfix):
32+

Resultado: 5.00
----------------------------
(teclado:)
Expressao (ints de 1 digito):
5^2 *( 3 +2)

Expressao (Postfix):
52^32+*

Resultado: 125.00
```

SAÍDA