Parte 5

Os argumentos de `main`

O cabeçalho completo da função `main` é:

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

Os argumentos `argc` e `argv` desempenham o mesmo papel que o argumento `args` na declaração do método `main` em Java:

```
public static void main(String[] args)
```

Através de `argc` e de `argv` é possível aceder aos argumentos usados na invocação do programa.

- O valor de `argc` é o número de argumentos dados ao programa mais 1. Se o seu valor for 1, o programa foi chamado sem qualquer argumento.
- `argv` é um *array* de *strings* com `argc` elementos. Em `argv[0]` encontra-se o nome através do qual o programa foi invocado, e nas posições de `1` a `argc-1` encontram-se os argumentos do programa. Na posição `argc`, `argv` contém o valor `NULL`.

Se o programa for executado pelo comando

```
./o-meu-programa 1o-arg "o segundo" 3 olá mundo
```

`argc` terá o valor 6 e `argv` será

```
argv[] = {
    "./o-meu-programa",
    "1o-arg",
    "o segundo",
    "3",
    "olá",
    "mundo",
    NULL
}
```

O valor devolvido por `main`

O valor devolvido pela função `main` é um inteiro e é usado para comunicar, a quem executa o programa, como correu a sua execução. A convenção usada é:

- Para indicar que a execução decorreu sem problemas, a função `main` devolve o valor 0 (zero).
- Para indicar que ocorreu algum problema durante a execução do programa, a função `main` devolve um valor diferente de zero.

Experimente criar o programa `ok`, que devolve 0, o programa `not-ok`, que devolve 1, e executar os comandos:

```
if ./ok; then echo Ok; else echo 'Not ok'; fi
```

e

```
if ./not-ok; then echo Ok; else echo 'Not ok'; fi
```

A função `fgets`

O protótipo desta função, contido em `stdio.h`, é

```
char *fgets(char *s, int size, FILE *stream);
```

`fgets` lê uma linha de texto e coloca-a na zona de memória apontada por `s`, incluindo o carácter correspondente ao fim de linha `'\n'`, seguida do terminador `'\0'`. Se tudo isto ocupar mais do que `size`*bytes*, só lê `size-1` *bytes* e coloca-os lá seguidos do terminador. (Uma chamada subsequente a esta ou a outra função de leitura, sobre o mesmo ficheiro, continuará a leitura a partir do ponto em que a anterior a tenha interrompido.)

O argumento `stream` indica o ficheiro de onde será feita a leitura e corresponderá a um valor devolvido, por exemplo, pela função fopen.

Se não se pretender ler de um ficheiro (para ler texto introduzido manualmente, por exemplo), o valor de `stream` deverá ser `stdin`, como no esquema de código seguinte:

```
#define MAXLINHA 4096

...

{
  char linha[MAXLINHA];

  ...

  ... fgets(linha, MAXLINHA, stdin) ...

  ...

}
```

`fgets` devolve `NULL` se o fim do ficheiro já foi atingido e não foi possível ler nada, e `s` (ie, o 1º argumento) se leu alguma coisa. Para indicar o fim dos dados quando se lê de `stdin`, prime-se `C-d`.

Tópicos relacionados: as funções (e macros) fgetc, getc, getchar, fputs, fputc, putc, putchar e puts.

A função `getline`

(Esta função não pertence ao standard ISO C, mas ao standard POSIX, e pode não ser suportada em todos os sistemas ou compiladores.)

O protótipo desta função, contido em `stdio.h`, é

```
ssize_t getline(char **lineptr, size_t *n, FILE *stream);
```

A função `getline` faz o mesmo que `fgets`, com duas diferenças. A primeira, e maior, diferença é que pode aumentar o tamanho da zona de memória que lhe é passada, para garantir que há espaço para a linha lida. A segunda diferença reside no valor que é devolvido, que, neste caso, é o comprimento da linha lida, incluindo o fim de linha. (O valor devolvido será -1 se a função não conseguir ler uma linha.)

O primeiro argumento da função contém o *endereço* de uma variável que contém o *endereço* da zona de memória onde colocar o conteúdo da linha lida. O segundo argumento contém o *endereço* de uma variável que contém o tamanho dessa zona de memória. Esta zona de memória deverá ter sido obtida através da função `malloc`.

Se o valor de `*lineptr` for `NULL`, ie, se a chamada não indicar a zona de destino da linha lida, `getline` reserva uma zona de memória com o tamanho necessário (através de `malloc`), e coloca o seu endereço em `*lineptr` e o seu tamanho em `*n`.

Se o valor de `*lineptr` não for `NULL` e o tamanho indicado em `*n`, na altura da chamada, não for suficiente para albergar a linha, `getline` reserva uma nova zona de memória com a dimensão necessária (através da função `realloc`, que liberta também a zona de memória antiga), e coloca o novo endereço em `*lineptr` e a nova dimensão em `*n`.

Os valores de `lineptr` e de `n` têm de ser endereços válidos, ie, diferentes de `NULL`.

O endereço contido em `*lineptr` depois de a função ter terminado deverá ser libertado pelo programa, quando essa memória deixar de ser necessária.