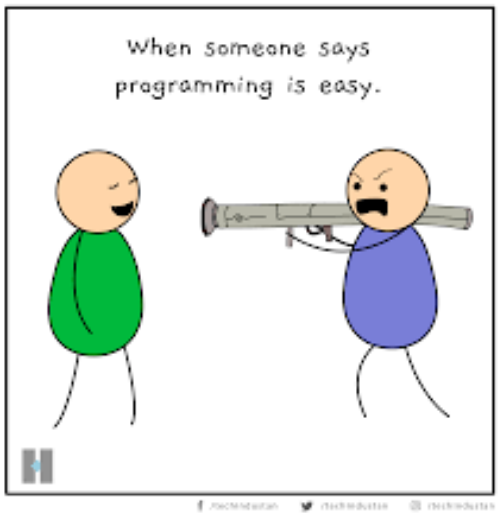
Meia dúzia de problemas introdutórios



2020/2021

Preparativos

Em Programação Imperativa vamos aprender a programar com C.

Em C, como em muitas linguagens, um programa é essencialmente uma coleção de funções. Entre elas há uma função especial, a função `main`. Esta função `main` é especial porque é pela função `main` que o programa começa as operações, quando o mandamos correr. Inversamente, quando a função `main` esgota as suas instruções, o programa termina.

A linguagem C é uma linguagem relativamente antiga e quando foi desenhada, há quase 50 anos, a informática e a programação eram bastante diferentes do que são hoje. Por isso, não é de admirar que, comparativamente com outras linguagens mais recentes, o C pareça chato, picuinhas e antiquado. Na verdade, o C é chato, picuinhas e antiquado mesmo (habitue-se…), mas continua a ser uma linguagem fantástica, que nós, programadores, veneramos acima de todas as outras, e com a qual podemos aprender imenso.

Para programar com C, primeiro precisamos de um computador. Depois, precisamos que nesse computador exista um editor de texto, com o qual escrevemos

os programas. Finalmente, precisamos de um compilador da linguagem C, para transformar os nossos programas-fonte, isto é, os programas que escrevemos no editor de texto, em programas executáveis no nosso computador.

O editor de texto "oficial" em Programação Imperativa é o Sublime Text. É muito bom e popular e tem a vantagem de existir para Windows, MacOS e Linux. Se ainda não o fez, instale o Sublime Text no seu computador.

O compilador "oficial" em Programação Imperativa é o chamado `gcc`. Existe para Windows, MacOS e Linux. Aliás, faz parte destes dois últimos sistemas operativos e já vem instalado. Em Windows, é preciso instalá-lo.

Começando a programar

Programar é escrever programas usando uma linguagem de programação.

Talvez você ainda se lembre como foi que aprendeu a escrever, na escola básica. Primeiro aprendeu a desenhar algumas letras: 'a', 'o', 'p', 'm', 't'. Depois, aprendeu a juntar letras para formar sílabas: 'p' mais 'a' dá "pa"; 't' mais 'o' dá "to". A seguir, justapondo letras ou letras e sílabas, aprendeu a escrever palavras: "pato". Passado algum tempo, conseguiu uma coisa fantástica: juntar palavras para formar uma frase que representa uma ideia: "o pato sabe nadar". Mais tarde, organizou uma série de ideias na sua cabeça e redigiu uma composição, encadeando por escrito as frases que correspondem a essas ideias.

Para a aprender a programar, podíamos usar a mesma abordagem, mas não. Vamos começar ao contrário: partimos de uma composição que alguém terá preparado, e vamos analisá-la, para ver como se expressam as ideias, como são compostas as frases, quais são as palavras usadas e qual será o significado daquilo tudo.

No nosso caso, a composição é um programa em C, bem entendido, o qual realiza uma determinada tarefa.

Começamos com um programa simples, que estudámos nas aulas teóricas: um programa que soma dois números:

```
// Sum, basic version
```

```c
#include <stdio.h>

int main(void)
{
  int x;
  int y;
  scanf("%d%d", &x, &y);
  int z = x + y;
  printf("%d\n", z);
  return 0;
}
```

É o mais simples que se pode arranjar: um programa que faz umas contas elementares e que é formado apenas pela função `main`.

Antes de começarmos a trabalhar, devemos preparar a nossa diretoria de trabalho.

Diretoria de trabalho

Se ainda não o fez, crie na diretoria `Documents` (ou noutra com a mesma vocação) uma subdiretoria para os seus trabalhos na nossa cadeira, chamada `PI_2021`.

Atenção: em geral, não é boa ideia dar às diretorias que contêm programas nomes formados por várias palavras (isto é, várias palavras separadas por espaços) ou com letras com sinais diacríticos, pois certos compiladores dão-se mal com isso. Aliás, para maior tranquilidade, evite que na hierarquia de diretorias que levam às suas áreas de trabalho de programação aparecem diretorias com nomes com espaços ou caracteres esquisitos.

Crie na diretoria de trabalho uma subdiretoria `sources` e outra `work`, ao lado de outras diretorias que lá existam.

Na nossa cadeira escreveremos algumas dezenas de programas. Em princípio, ficarão todos nesta diretoria `sources`. Portanto, é uma diretoria preciosa. Faça *backup* dela frequentemente. Faça *backup* também da diretoria PI_2021 de vez em quando.

A diretoria `work` servirá para guardar os ficheiros de dados que os nossos programas usarão. Por enquanto fica vazia.

Sugiro que crie outras diretorias dentro da área de trabalho, por exemplo, `aulas`, para as apresentações das aulas, `docs`, para textos fornecidos pelos professores (por exemplo, o presente guião), etc.

Escrever o programa no editor

Está na altura de copiar o programa acima para o editor de texto. É melhor copiar teclando do que copiar com *copy-paste*. Ao teclar, educamos os nossos dedos e prestamos atenção às diversas partes do programa. Copiar com `copy-paste` é mais rápido, mas aprende-se menos!

Guarde-o na diretoria `sources`, num ficheiro novo com um nome à sua escolha, mas que seja fácil de localizar, com extensão ".c", por exemplo, `sum.c`.

Compilação

Os programas são feitos para *correr*. A tarefa que o nosso programa realiza não é fantástica, mas, pronto, corramo-lo.

Tratando-se de um programa em C, antes de o correr temos de o compilar.

Começamos por abrir uma janela de comando, ou consola, na diretoria `sources`.

Em Windows, isso faz-se diretamente, assim: no *Windows Explorer* clica-se na pasta `sources` carregando, ao mesmo tempo, na tecla *Shift* e escolhe-se `Open command window here`. Ou então abre-se uma janela com a pasta `sources` e na caixa do endereço escreve-se `cmd`.

Em MacOS, é parecido: no *Finder*, seleciona-se a pasta onde queremos abrir a janela e fazemos `Finder->Services->New Terminal at Folder`.

Arrume o seu *desktop* de maneira a mostrar em permanência pelo menos três janelas: a do navegador que contém o presente guião; a do editor de texto, que contém o programa; e a janela de comando. Dimensione bem as janelas, de ma-

neira que elas não se tapem umas às outras, para poder trabalhar confortavelmente.

Admitindo que o nome do ficheiro é `sum.c`, para compilar, damos o seguinte comando, na consola:

```
$ gcc –Wall sum.c
```

Nesta linha, o cifrão representa o *pronto* que está em vigor na janela de comando. No seu computador, o pronto poderá ser outro.

Da primeira vez que dá este comando no Windows, pode surgir uma mensagem de erro, indicando que o Windows "não conhece o `gcc`". O que isto significa é que a diretoria onde está o `gcc` (em princípio a diretoria `C:\MinGW\bin`), não figura no *path*, isto é, na lista de diretorias onde o Windows procura os comandos que nós damos.

Para colocar a diretoria `C:\MinGW\bin` no *path* dê o seguinte comando:

```
$ setx PATH "%PATH%;C:\MinGW\bin"
```

Confira, saindo da janela de comando, voltando a abrir e dando o seguinte comando:

```
$ path
```

Notará que a diretoria `C:MinGW\bin` surge no fim da lista de diretorias do *path*.

Compile de novo, com o comando `gcc`:

```
 $ gcc –Wall sum.c
```

Em Windows, nada acontece na consola, mas surge na diretoria um ficheiro `a.exe` que contém o programa executável. Em MacOS ou Linux é análogo, mas o ficheiro criado chama-se `a.out`.

Se a compilação tivesse corrido mal, surgiriam mensagens de erro na janela de comando.

Execução

Para executar o programa, invocamos na linha de comando o ficheiro `a.exe` ou `a.out`, consoante o sistema operativo, Windows ou MacOS/Linux.

Eis a transcrição de uma sessão em Windows, com três ensaios:

```
>a.exe
5 7
12

>a.exe
100 200
300

>a.exe
571 219
790
>
```

Note bem: em cada caso, a seguir ao comando `a.exe`, o programa começa a correr e fica à espera que nós dêmos dois números. Na primeira experiência demos 5 e 7, na segunda 100 e 200 e na terceira 571 e 219. O programa respondeu, escrevendo 12, 300 e 790, respetivamente.

E agora a mesma coisa, em MacOS:

```
$ ./a.out
5 7
12
$ ./a.out
100 200
300
$ ./a.out
571 219
790
$
```

Submissão

Alguns dos programas que você fará ao longo da cadeira são avaliados automaticamente pelo Mooshak. Nesses casos, o professor terá colocado no Mooshak vários casos de teste. O seu programa correrá com cada um dos casos de teste. Se

em todos os casos o resultado do seu programa for o esperado, o seu programa será considerado *aceite* pelo Mooshak.

Para praticar este funcionamento, submetamos o programa `sum.c` no concurso PI_2021_P1 do Mooshak.

Se ainda não tiver conta no Mooshak, por favor crie uma, seguindo as instruções na página Web da cadeira (na tutoria).

Ao fazer *login* no Mooshak, escolha o concurso PI_2021_P1.

Constatará que o concurso tem seis problemas, A, B, C, D, E e F. O problema A é o problema que deu origem ao programa `sum.c`. Os outros problemas vêm a seguir.

Use a interface do Mooshak para submeter o seu programa, no problema A.

Se não se tiver perdido no caminho até aqui, uns segundos após a submissão, poderá ver na listagem que o seu programa está *Accepted*!

Agora que o programa foi aceite, não lhe mexa mais! Se precisar de fazer modificações, para acrescentar novas funcionalidades, trabalhe a partir de uma cópia.

Em todo o caso, se perder o ficheiro de um programa que já foi aceite pelo Mooshak, pode sempre ir buscá-lo ao Mooshak, de novo.

Note que, noutros problemas, mais adiante, será normal continuar a trabalhar no mesmo programa, para introduzir novas funcionalidades. Convém, é claro, não modificar aquelas que já foram validadas.

Analisando o programa

Este nosso primeiro programa, na verdade, foi "oferecido" já pronto.

Para ganhar familiaridade com a linguagem, modifique o programa (sempre a partir de uma cópia) de maneira a, por exemplo, somar 3 números, realizar outras operações aritméticas, etc.

Experimente também mudar o programa, de maneira a forçar alguns erros. Leia as mensagens de erro e interprete-as. Há de reparar que nem sempre as mensagens de erro descrevem o erro convenientemente.

Por exemplo, apague a diretiva `#include` na primeira linha. Que acontece? Apague, o `void` na lista de argumentos da função `main`? Apague o `return 0;` na função `main`. Acrescente um `return 0;`Omita o ponto e vírgula no `return 0;`. Apague o `&` dentro do `scanf`. Apague a linha `int x;`. Apague o sinal `=` na linha `z = x + y;` No `printf`, escreva `%f`, em vez de `%d`. E depois `%o`. E também `%x`. E ainda `%y`. Omita o `\n`. Escreva `som` em vez de `sum`, na chamada da função. Experimente escrever `Int` em vez de `int`. Apague o `int` antes do `z`, etc.

Em cada caso, interprete a mensagem de erro do compilador, se houver. Às vezes há erros — em inglês, *errors* — e também avisos — em inglês *warnings*. Nós só ficamos satisfeitos quando não houver nem erros, nem avisos. Em rigor, um programa sem erros correrá, mesmo que tenha avisos, mas não é recomendável. Se correr um programa com erros, na verdade estará a correr a anterior versão do programa que terá sido compilada sem erros, e não aquela que tinha erros.

Não havendo erros nem avisos, como se comporta o programa executável? Bem? Nem por isso? Mal? Não faça apenas estas experiências: tome nota das suas conclusões no seu caderno (ou num documento no seu computador), ou, pelo menos, na sua memória.

Repare que será este o ambiente de trabalho nos primeiros tempos: uma janela com o editor, que usamos para escrever o programa, e uma janela de comando, que usamos para compilar e correr o programa.

Segundo problema

O primeiro problema — somar dois números — não constituiu verdadeiramente um problema, já que foi sobejamente tratado nas aulas teóricas. Este agora é um verdadeiro problema: dado um número inteiro positivo, calcular o maior número inteiro que multiplicado pelo número dado não causa *overflow*. Uma operação

de multiplicação causará *overflow* se o produto for maior que o maior número inteiro representável.

Escreva um programa novo para realizar este cálculo. Bastará modificar ligeiramente o programa anterior.

Quando estiver confiante que o seu programa calcula bem, submeta-o no Mooshak, problema B.

Terceiro problema

Recorde o programa da soma elementar, aquela que somava só recorrendo às operações elementares "mais 1", "menos 1" e "igual a zero":

```
// Sum, recursive, using only elementary operations

#include <stdio.h>

int sum(int x, int y)
{
  return y == 0 ? x : sum(x+1, y-1);
}

int main(void)
{
  int x;
  int y;
  scanf("%d%d", &x, &y);
  int z = sum(x, y);
  printf("%d\n", z);
  return 0;
}
```

Acrescente a este programa (isto é, a uma cópia deste programa), uma função `twice`, que, dado um número calcula o seu dobro, recorrendo apenas às operações elementares ("mais 1", "menos 1" e "igual a zero") e à função `sum` (não necessariamente a todas elas…) Ajuste a função `main`, para agora chamar a função `twice` (em vez da função `sum`).

Se não fosse a restrição de que neste exercício apenas queremos usar as operações elementares e a função `sum`, programaríamos a função `twice` assim:

```
int twice(int x)
{
  return 2*x;
}
```

Mas esta função não vale, pois neste exercício é proibido usar a multiplicação. O que é preciso é substituir a instrução `return 2*x;` por uma outra, equivalente, mas que respeite as restrições do exercício.

Quando estiver confiante que o seu programa calcula bem, submeta-o no Mooshak, problema C.

Quarto problema

Ainda usando só as operações elementares, programe uma função, `half` que, dado um número positivo, calcula a parte inteira de metade desse número. Repare, se o número for 0 ou o número menos um for 0, então o resultado é zero; caso contrário, subtraímos duas vezes 1 ao número, calculamos metade disso e somamos um ao resultado.

Se não fossem as restrições, programaríamos assim:

```
int half(int x)
{
  return x/2;
}
```

Ora, isto não é aceitável, neste exercício. Queremos uma formulação que só recorra às operações elementares.

Submeta-o no Mooshak, problema D. Note bem, o seu programa só será validado se apenas usar as três operações elementares.

Quinto problema

Para este quinto problema inspire-se no programa da soma de números inteiros consecutivos estudado na aula teórica.

Escreva um programa que calcula a soma dos quadrados de todos os números inteiros entre dois números dados. Por exemplo, se os números dados for 5 e 7, o

programa escreverá 110, pois 110 é igual a 25 + 36 + 49. Em todos os casos, o primeiro número é menor ou igual ao segundo.

Neste caso, precisamos de uma função com dois argumentos, cujo cabeçalho é análogo ao da função `sum`, que entrou no terceiro problema.

No corpo da função pode usar as operações aritméticas do C, normalmente. Já não há restrições. Tal como no caso visto na aula, o que interessa é programar a "fórmula" que resolve a questão…

De novo, quando estiver confiante que o seu programa calcula bem, submeta-o no Mooshak, problema E.

Sexto problema

Queremos um programa que aceite dois números inteiros positivos e que calcule o número cuja representação decimal é a concatenação das representações decimais dos argumentos.

Este problema, tal como o anterior, requer uma função com dois argumentos. Se a função se chamar `concatenate` então, por exemplo, `concatenate(34, 712)` vale `34712`, `concatenate(1, 4)` vale `14`, `concatenate(1000, 43)` vale `100043`.

Note que, por hipótese, ambos os argumentos são números positivos.

Submeta o seu programa no problema F.