Ciências ULisboa

Programação

Enunciado do trabalho — Ano lectivo 2021-22

Grupos de **3** alunos — Cotação: 6 valores

Data-limite de entrega: 17 de Janeiro às 23h59m

Instruções para a entrega:

- Entregar apenas este ficheiro: **vida.c**
  [com informação extra no nome do ficheiro, explicado abaixo]
- O ficheiro deve estar identificado nas linhas iniciais com o número do grupo e com o número e nome de cada um dos elementos do grupo; essa identificação deve ser feita usando comentários delimitados por /* ... */.
- Antes de entregar o ficheiro, mudar o seu nome para **vida_XX_fcXXXXX.c**
  onde **XX** é o *número do grupo* e **fcXXXXX** é a referência ao *número do aluno que submete o trabalho no moodle*.
  - exemplo: **`vida_09_fc90900.c`** se for o aluno 90900 a submeter o trabalho no moodle, e o seu grupo for o nº 9
- Fazer upload do ficheiro no moodle, quando a página de entrega estiver aberta.

Requisitos do programa:

Assegure-se de que **vida.c** *(após editado por si)* compila correctamente, e de que o programa de teste que resulta de compilar **testa_vida.c** juntamente com **vida.c** executa correctamente no ambiente Windows padrão em que a disciplina foi leccionada.

Todas as compilações têm de usar as opções **`-ansi -Wall`** do compilador **gcc**, não originando erros nem avisos ("warnings").

As funções serão avaliadas pela correcção, mas também pela estrutura do código e pela formatação. Não deverá ultrapassar as 80 colunas por linha. Comente o seu código onde isso for útil.
*A reutilização de certas funções ao serem invocadas dentro de outras funções (não apenas na **main**) é valorizada.*

Tema — criar vida num computador

Como se fôssemos deuses, vamos criar vida. Mas *in silico*. É portanto vida artificial.

Vamos escrever um programa em C que concretiza o algoritmo Jogo da Vida, um *autómato celular* proposto por John Conway em 1968.

Este algoritmo é muito famoso e existe muita informação sobre ele na web. Pesquise para se informar melhor. Um bom ponto de partida é
https://en.wikipedia.org/wiki/Conway's_Game_of_Life

Existem muitas implementações em C e noutras linguagens de programação. Mas vamos definir uma versão original com contratos próprios, que tem de passar testes específicos.

Autómatos celulares

Numa rede com uma grelha regular, cada nó recebe input dos seus vizinhos mais próximos e actualiza o seu estado dependendo desse input. Usualmente considera-se grelhas com uma simetria como a das grelhas ilustradas parcialmente abaixo.

Dois exemplos de padrões de conectividade:

Jogo da Vida (ou *Computador Vida*): topologia e modo de actualização

- Decorre numa grelha regular, localmente quadrada. Globalmente, o *mundo* é um rectângulo com tamanho configurável.
- Os nós são chamados de *células*. Cada célula está *viva* (estado 1) ou *morta* (estado 0).
- Cada célula recebe input de 8 vizinhos (dos lados e dos cantos) e também dela própria.
- A actualização do estado das células é *síncrona*, ou seja, todas as células do mundo se actualizam em simultâneo, de cada vez que o tempo (discreto) avança 1 unidade.

Jogo da Vida: regras de actualização das células

Começamos por definir o conjunto de regras mais conhecido, e que o nosso programa vai assumir por omissão. No entanto, é possível usar outras regras.

- Se uma célula está viva:
  - → sobreviverá na geração seguinte se tiver 2 ou 3 vizinhos vivos;
  - → morrerá se tiver mais de 3 vizinhos vivos (“sobrelotação”);
  - → morrerá se tiver menos de 2 vizinhos vivos (“desprotecção”).

- Se uma célula está morta:
  - → nascerá na geração seguinte se tiver exactamente 3 vizinhos vivos;
  - → permanecerá morta se não tiver exactamente 3 vizinhos vivos.

Jogo da Vida: dinâmica

A partir de uma *condição inicial* do mundo (estado inicial das células), o mundo evolui ao longo de gerações sucessivas. Cabe ao utilizador do programa:

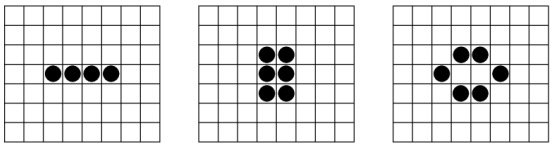
- fixar as regras de evolução (tipicamente, as descritas acima);
- inicializar o mundo, i.e., decidir o estado inicial de cada célula;
- iterar o mundo durante o número de iterações ("gerações") desejado.

Portanto, o utilizador só intervém no início da simulação. Eventualmente pode intervir a meio da simulação, interrompendo o programa caso não deseje levar a simulação até ao fim.

Jogo da Vida: vida

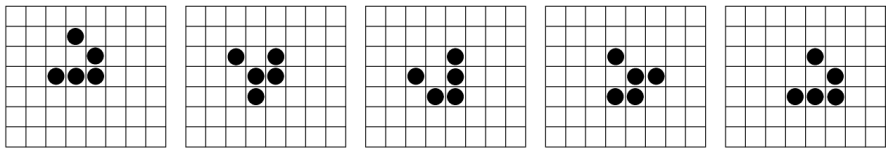
"Seres" nascem, crescem, evoluem, deslocam-se, auto-reproduzem-se até ao infinito, …

Exemplo 1: convergência para uma forma estacionária



geração 0 geração 1 geração 2 e seguintes

Exemplo 2: o patinador

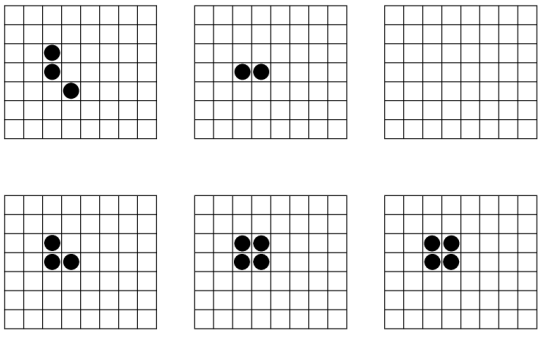


geração 0 geração 1 geração 2 geração 3 geração 4

Após 4 gerações, o patinador moveu-se 1 casa ao longo da diagonal.

Exemplo 3: morte versus sobrevivência

Ilustra-se duas evoluções independentes, com condições iniciais ligeiramente diferentes:



geração 0 geração 1 geração 2 e seguintes

São disponibilizados aos alunos outros exemplos de condições iniciais, que podem ser testados uma vez completado o programa. Ver a pasta `mundos_iniciais`.

Para testar rapidamente diversas configurações de maneira independente deste trabalho, a seguinte página web é muito útil:

https://dev.to/lexjacobs/conways-game-of-life-with-different-rules-13l0

mais concretamente o simulador proposto pelo autor:

https://conways-game-of-life-explorer.herokuapp.com/

Este simulador permite definir o tamanho do mundo, inicializar facilmente as células, alterar as regras de actualização das células, bem como configurar outros detalhes da simulação.

Concretização num programa em C

Regra de actualização das células

Por conveniência de programação, condensamos as regras descritas acima no triplo

regra = {*max_vizinhos*, *min_vizinhos*, *vizinhos_para_nascer*}

com esta interpretação:

- uma célula viva morrerá por sobrelotação se tiver mais de *max_vizinhos* vizinhos vivos;
- uma célula viva morrerá por desprotecção se tiver menos de *min_vizinhos* vizinhos vivos;
- uma célula morta nascerá se tiver exactamente *vizinhos_para_nascer* vizinhos vivos.

Assim, a regra padrão do Jogo da Vida é o triplo {3, 2, 3}.

Porém, nada nos impede de experimentar outras regras, dentro dos valores admissíveis para os parâmetros (ver contratos das funções).

Representação interna do estado das células e do mundo

Como sugerido acima, uma célula viva é representada internamente por 1 e uma célula morta por 0.

O mundo é representado como uma matriz rectangular. O tamanho *numLinhas* × *numColunas* da matriz *virtual* é um parâmetro da simulação, definido pelo utilizador.

Mais concretamente, representamos o mundo como um vector bidimensional (matriz) em que cada elemento é 0 ou 1.

Na nossa implementação em C,

- a matriz *virtual* mapeia exactamente o mundo do Jogo da Vida;
- a matriz *efectiva* tem 2 linhas e 2 colunas a mais que a matriz virtual; portanto, tem tamanho (*numLinhas* + 2) × (*numColunas* + 2);
- a matriz virtual está dentro da matriz efectiva, e ambas estão dentro de uma matriz suficientemente grande para as conter, e criada em tempo de compilação; chamemos-lhe matriz *real*;
- por limitações do ANSI C, o número de colunas da matriz real tem de ser decidido à partida, sendo usado na **main** e em todas as funções que processam matrizes com mundos; esse número é dado por **`COLUNAS_MAX`** + 2;

- assim, o número de colunas *numColunas* da matriz virtual (e portanto do mundo simulado) não pode ultrapassar **`COLUNAS_MAX`**;
- na matriz efectiva, as linhas de índices 0 e (*numLinhas* + 1) são usadas para cálculos auxiliares; as colunas de índices 0 e (*numColunas* + 1) são também usadas para cálculos auxiliares; a explicação é dada mais abaixo e tem a ver com as condições-fronteira do mundo;
- portanto, a matriz virtual começa na linha de índice 1 da matriz real, e na coluna de índice 1 da mesma matriz.

Representação externa do estado das células e do mundo

A representação externa é a que é usada:

- para mostrar, imprimindo no ecrã, o estado do mundo;
- para guardar, num ficheiro de texto, o estado do mundo numa certa iteração;
- para inicializar facilmente, a partir de um ficheiro de texto, o estado do mundo.

Optamos por representar

- uma célula morta (representação interna = 0) pela string **`". "`**
- uma célula viva (representação interna = 1) pela string **`"X "`**

As linhas da matriz são impressas consecutivamente.

Exemplo de padrão de células mortas e células vivas num mundo 11 × 11:

```
. . . . . . . . . . .
. . . . . . . . . . .
. . . . . X . . . . .
. . . . X X X . . . .
. . . X . X . X . . .
. . X X X . X X X . .
. . . X . X . X . . .
. . . . X X X . . . .
. . . . . X . . . . .
. . . . . . . . . . .
. . . . . . . . . . .
```

Condições-fronteira

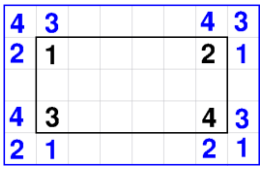
O que acontece à dinâmica perto dos limites do mundo? No exemplo 2 acima, o que acontece ao patinador quando atinge a fronteira? Mais em geral, para as células que estão nas linhas ou colunas limítrofes do mundo simulado, quais são os vizinhos que elas "vêem"?

Na literatura sobre autómatos celulares, há várias soluções propostas. Estas incluem mundos infinitos (sem fronteiras!), e mundos com condições-fronteira "reflectidas" (em que uma célula substitui um vizinho inexistente do lado de "fora" do mundo por um vizinho contíguo no interior do mundo, cuja posição é "reflectida" tendo como eixo de reflexão a célula a actualizar).

Neste trabalho vamos considerar outras condições-fronteira, conhecidas como ***periódicas***:

o mundo "dobra-se" sobre si mesmo de tal maneira que

- cada célula da última linha tem como vizinhos "de baixo" as células nas posições "contíguas", mas na primeira linha, e vice-versa;
- cada célula da última coluna tem como vizinhos "à direita" as células nas posições "contíguas", mas na primeira coluna, e vice-versa;
- os cantos suscitam uma análise mais cuidada; a figura abaixo ilustra os vizinhos que também são cantos (numerados a **azul**) que cada canto do mundo (numerado a **preto**) "vê" como se estivessem ao seu lado:



Lembrando a explicação anterior, a matriz ***virtual*** corresponde à parte a preto da figura, e a matriz ***efectiva*** inclui a matriz virtual mais a parte a azul.

Estratégia de concretização num programa em C

A actualização do mundo é a parte mais importante do trabalho. Especial atenção deve ser dada às células na fronteira do mundo, para concretizar as condições-fronteira indicadas.

Na estratégia seguida neste trabalho, em cada iteração, preenche-se a parte ilustrada a azul da matriz efectiva, copiando para lá valores limítrofes da matriz virtual ilustrada a preto:

- coloca-se uma cópia da última coluna da matriz virtual na primeira coluna da matriz efectiva;
- coloca-se uma cópia da primeira coluna da matriz virtual na última coluna da matriz efectiva;
- faz-se o mesmo, respectivamente, para as primeira e última linhas da matriz virtual, notando que
  - a primeira linha da matriz virtual é a segunda linha da matriz efectiva;
  - a última linha da matriz virtual é a penúltima linha da matriz efectiva.

A partir daí, basta actualizar o "miolo" da matriz efectiva, ou seja, a matriz virtual que representa o mundo original, com cada célula a "ver" 8 células à sua volta e a poder aplicar uniformemente a regra de actualização individual.

⚠ A actualização das células tem de ser síncrona. Isso implica que não se pode alterar o estado de uma célula antes de calcular o próximo estado das células suas vizinhas.

Informaticamente, adiamos a actualização de cada célula até que o novo valor de todas as células esteja calculado. Usamos uma matriz auxiliar, com o mesmo tamanho da matriz principal, para guardar temporariamente os valores calculados de cada célula para a próxima iteração. Uma vez completado o cálculo da próxima geração do mundo, os valores são recopiados para a matriz principal.

Na função `iteraMundo`, a matriz auxiliar é passada através do parâmetro `mundoAuxiliar`.

Especificação e contratos

Leia atentamente os contratos das funções que estão no ficheiro **vida.h**.
O funcionamento das funções a desenvolver pelos alunos no ficheiro **vida.c**, que será explorado pela aplicação **testa_vida** e outras aplicações cliente de que **app_animacoes** é exemplo, é definido de diferentes modos que se complementam:

- leitura dos contratos acima referidos;
- descrição dada no enunciado do trabalho (este documento);
- senso comum.

A confiança no programa desenvolvido aumentará se as funções passarem com sucesso os testes propostos em **testa_vida**, ou outros testes adequados, criados pelos alunos. O objectivo principal dos testes é ajudar a verificar se o comportamento desejado está bem concretizado no *software*.

É importante para o sucesso do trabalho reduzir a ambiguidade na descrição do comportamento desejado. Os resultados de testes fornecidos *também* podem ser úteis para isso ("testes como documentação").

Trabalho a realizar pelos alunos

Há 2 partes, mas apenas a 1ª é entregue e avaliada. A 2ª parte é para treino dos alunos.

Parte 1

A primeira parte consiste em *definir* ("implementar") no ficheiro **vida.c** todas as funções *declaradas* em **vida.h**, e aqui listadas:

- `celulaVive`
- `zeraMundo`
- `atribuiValorCelula`
- `valorDaCelula`
- `mostraMundo`
- `escreveMundo`
- `leMundo`
- `iteraMundo`
- `iteraMundoNgeracoes`

Todas as funções a definir devem estar de acordo com a respectiva documentação, ou seja, ***é preciso respeitar os contratos.***

Pode definir outras funções auxiliares caso considere útil; mas as funções que já estão presentes são suficientes.
Espera-se que o trabalho faça um uso adequado de funções auxiliares. Será penalizado se não o fizer. Por exemplo:

- `iteraMundo` pode usar `celulaVive` como auxiliar
- `iteraMundoNgeracoes` pode (e deve!) usar `iteraMundo` como auxiliar

A função `iteraMundo` é a mais desafiante.

Mesmo que não consiga definir a função `iteraMundo`, pode apresentar uma definição de `iteraMundoNgeracoes` que use (correctamente) `iteraMundo` como auxiliar. Os docentes usarão uma versão funcional de `iteraMundo` como auxiliar ao testar a versão dos alunos de `iteraMundoNgeracoes`.

Testes da Parte 1

Os testes são opcionais, mas seria insensato não os fazer.

Se as funções estiverem correctamente definidas, a execução do programa de teste **testa_vida** fornecido aos alunos deve

- completar-se sem gerar qualquer erro em tempo de execução;
- enviar para o output standard (ecrã) texto igual ao que está registado no ficheiro **stdout-testa_vida.txt**; note que muitos dos testes são silenciosos, ou seja, não geram output caso a função esteja correcta;
- criar na pasta de trabalho o ficheiro **mundo_escrito.txt**.

Avaliação do output

Nas fases finais do trabalho, pode-se guardar em ficheiros de texto o output obtido e verificar se o mesmo já é igual ao pretendido, usando o comando **`FC`** (**File Compare**) numa linha de comandos do Windows ou **`fc.exe`** na PowerShell do Windows.

Exemplo de comandos para compilar e executar o trabalho, guardando o respectivo output:

```
C:\...\work> gcc -ansi -Wall -o testa_vida testa_vida.c vida.c
C:\...\work> .\testa_vida.exe > result.txt
```

A sintaxe do comando **`FC`** é aqui ilustrada com ficheiros de texto simplificados. Neste caso, **`output.txt`** e **`output_2.txt`** são iguais, mas **`output_3.txt`** difere numa linha.

```
C:\...\pasta_auxiliar> FC output.txt output_2.txt
Comparing files output.txt and OUTPUT_2.TXT
FC: no differences encountered
```

```
C:\...\pasta_auxiliar> FC output.txt output_3.txt
Comparing files output.txt and OUTPUT_3.TXT
***** output.txt
Um texto assim.
Um texto assim assim.

***** OUTPUT_3.TXT
Um texto assim.
Um texto assim assado.

*****
```

Nota sobre o formato do output

Uma vez que é examinado o *output* de um programa que depende de código escrito pelos alunos (apenas na parte 1), determina-se que

> *a resolução apresentada pelos alunos deve obedecer aos exemplos dados no ficheiro* ***stdout-testa_vida.txt****.*

Parte 2

Na segunda parte os alunos criam uma pequena aplicação que

- tem uma interacção *user-friendly* com o utilizador
- incluindo um menu de opções;
- explora as funcionalidades presentes em **vida.c**;
- adicionalmente, usa a funcionalidade de *animação* do mundo proporcionada pela função **`animaMundo`**, que está declarada em **animacoes.h** e definida em **animacoes.c**

A função **`animaMundo`** é fornecida aos alunos completamente definida. Porém, só pode ser usada em aplicações se estiverem concretizadas as definições das suas funções auxiliares que estão declaradas em **vida.h**.

A aplicação **app_animacoes** exemplifica o uso de **`animaMundo`**.

Os ficheiros **animacoes.h**, **animacoes.c** e **app_animacoes.c** são apenas para a Parte 2.
Não fazem parte do trabalho que os alunos têm de entregar.