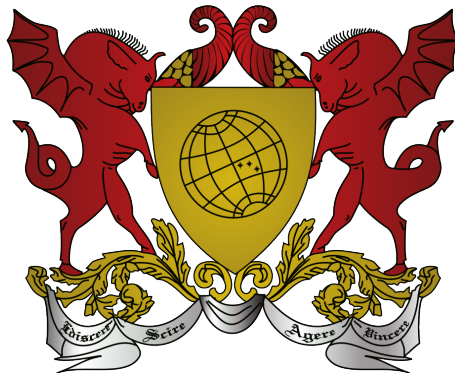


## **Gerador de imagens aleatórias**

Trabalho prático zero de projeto e análise de algoritmos - CCF 330F

**Mateus Pinto da Silva - 3489**

Trabalho apresentado para a  
graduação em Ciência da Computação



Departamento de ciências exatas  
Universidade Federal de Viçosa  
Brasil

## Sumário

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Pré-requisitos para a execução . . . . .	1
1.2	Como executar . . . . .	1
<b>2</b>	<b>Desenvolvimento</b>	<b>1</b>
2.1	O tipo abstrato de dados <i>paintingFrame</i> . . . . .	2
2.1.1	A estrutura . . . . .	2
2.1.2	A operação <i>paintingFrameCreateClearPaint</i> . . . . .	2
2.1.3	A operação <i>paintingFrameCreatePaintFromFile</i> . . . . .	2
2.1.4	A operação <i>paintingFrameInsertPaintInRandomPosition</i> . . . . .	3
2.1.5	A operação surpresa . . . . .	4
2.2	<i>auxFunctions</i> . . . . .	4
2.3	O <i>main.c</i> . . . . .	4
2.4	<i>makefile</i> . . . . .	4
<b>3</b>	<b>Referências</b>	<b>5</b>
<b>4</b>	<b>Conclusão</b>	<b>5</b>

## 1. Introdução

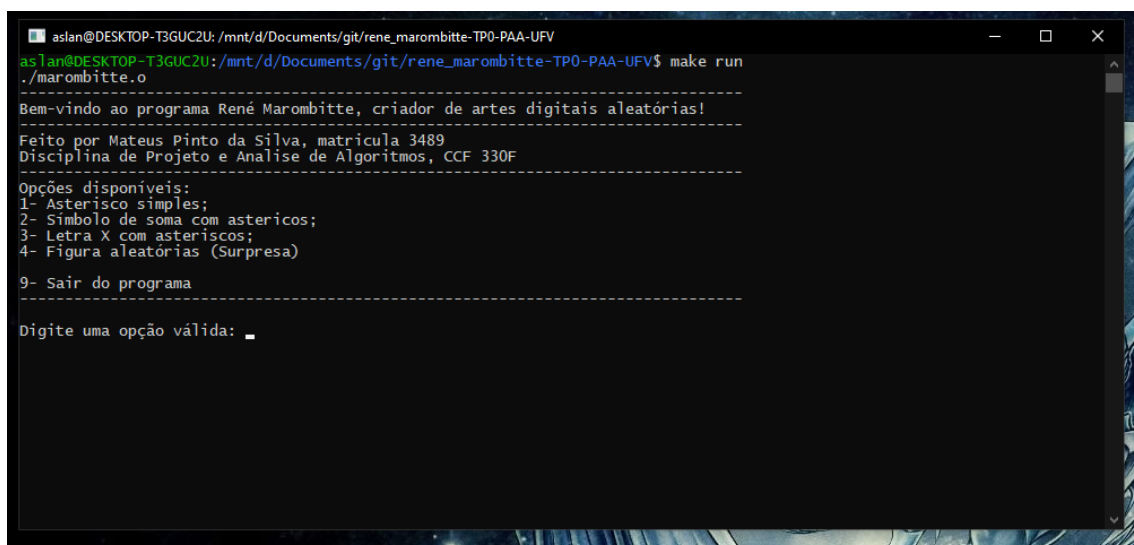
Este trabalho consiste em uma implementação de um programa que gera obras de arte aleatórias implementado em linguagem C, usando para isso números aleatórios, asteriscos, mais de um x, cruces e uma surpresa especialmente feita por mim. Tudo isso foi feito usando um tipo abstrato de dados criada de forma bastante genérica, permitindo seu reuso para outras aplicações, além de garantir eficiência e diminuição de erros usando alocação dinâmica em toda a estrutura.

### 1.1. Pré-requisitos para a execução

- Make
- GCC ou Clang (recomendável)
- Windows com WSL ou Linux (recomendável)

### 1.2. Como executar

1. Abra o terminal do Linux ou do WSL e digite “make” para compilar usando o GCC, ou “make clang” para compilar com o Clang (recomendável);
2. Digite “make run” para executar;
3. Siga o menu intuitivo do programa; e
4. Opcional: caso queira deletar o executável, digite “make clear”.



```
aslan@DESKTOP-T3GUC2U: /mnt/d/Documents/git/rene_marombitte-TP0-PAA-UFV
aslan@DESKTOP-T3GUC2U: /mnt/d/Documents/git/rene_marombitte-TP0-PAA-UFV$ make run
./marombitte.o
-----
Bem-vindo ao programa René Marombitte, criador de artes digitais aleatórias!
-----
Feito por Mateus Pinto da Silva, matrícula 3489
Disciplina de Projeto e Análise de Algoritmos, CCF 330F
-----
Opções disponíveis:
1- Asterisco simples;
2- Símbolo de soma com asteriscos;
3- Letra X com asteriscos;
4- Figura aleatórias (Surpresa)
9- Sair do programa
-----
Digite uma opção válida: _
```

Figura 1. A interface do programa.

## 2. Desenvolvimento

Todo o trabalho foi desenvolvido usando técnicas de encapsulamento para evitar erros, por isso foi criado o tipo abstrato de dados *paintingFrame*, além de um conjunto de funções chamada *auxFunctions*, que retira a necessidade de código excessivo no *main*. Além disso, preferiu-se usar leitura de arquivo para o carregamento dos desenhos (desde os mais simples, como os asteriscos, até os da operação surpresa). Assim, foi possível evitar o trabalho com posições de forma explícita.

## 2.1. O tipo abstrato de dados *paintingFrame*

Esse tipo abstrato de dados é uma abstração de uma pintura/desenho/quadro, contendo as operações de ler um desenho de um arquivo, exibi-lo na tela do computador, copia-lo de um *paintingFrame* para outro em local específico ou aleatório, dentre outros. Essa implementação foi a peça-chave para a realização do trabalho.

### 2.1.1. A estrutura

A estrutura é basicamente uma matriz dinamicamente alocada de caracteres, além da altura e do comprimento dela.

```
1 typedef struct paintingFrame
2 {
3     char ** pixels;
4     unsigned short int height;
5     unsigned short int width;
6 } paintingFrame;
```

### 2.1.2. A operação *paintingFrameCreateClearPaint*

Cria um desenho vazio. Essa operação utiliza outras três muito básicas, retirando da memória um possível outro desenho alocado, apagando sua possível referência de memória e alocando memória para a nova gravura, além de preenchê-la inteira com ' \0', equivalente a espaços vazios. Essa operação é chamada várias vezes em outros momentos.

```
1 paintingFrameDestroyPaint(instance);
2 paintingFrameStartPaint(instance);
3 return paintingFrameClearPaint(instance, height, width);
```

### 2.1.3. A operação *paintingFrameCreatePaintFromFile*

Cria um desenho básico na memória, lendo-o de um arquivo. Isso evita a necessidade de referenciar os caracteres usando posições, por exemplo. Todos os pontos finais no desenho são substituídos pelos ' \0', permitindo que outro desenho possa ocupar esse espaço, diferente do caractere espaço, que não pode ser substituído, o que é útil para criar desenhos com espaço vazio, como círculos. A primeira linha do arquivo de texto contém a altura seguido da largura do desenho.

```
1 fscanf(file, "%c", &swap);
2
3     if (swap != ' .')
4     {
5         ((**instance).pixels[i][j]) = swap;
6     }
```

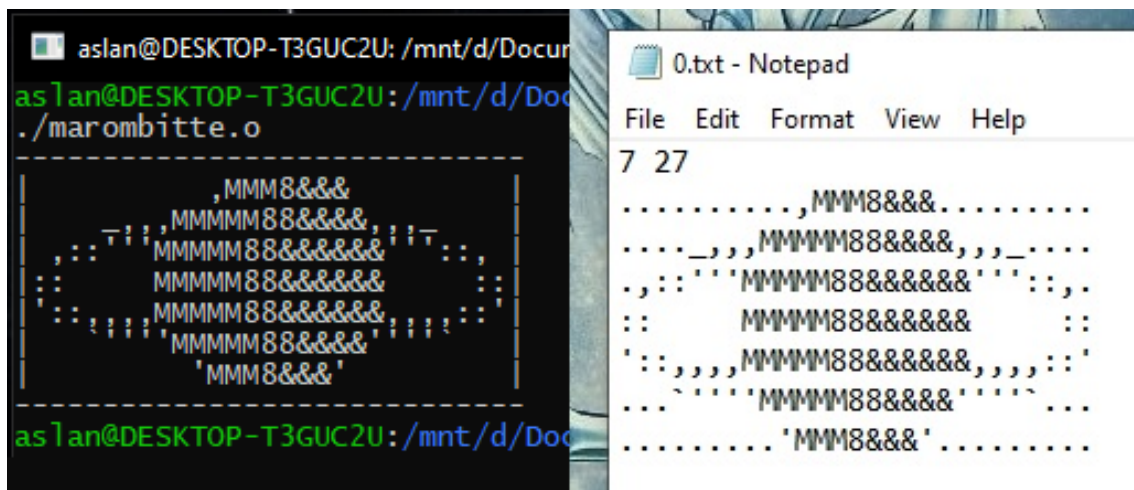


Figura 2. Desenho aberto de um arquivo de texto usando a operação. Note que o espaço entre o anel e o corpo do planeta não pode ser preenchido, porém o espaço externo ao anel pode.

#### 2.1.4. A operação *paintingFrameInsertPaintInRandomPosition*

Copia um desenho para um outro maior ou igual numa posição aleatória, tentando encontrar um espaço disponível para o desenho em que não haja sobreposição, sorteando novamente posições caso a primeira já esteja ocupada. Funciona usando cálculo de quantidade de caracteres na vertical e na horizontal.

A função aleatória foi implementada utilizando `rand()` e `srand()`. A primeira cria um número inteiro aleatório que pode gerar qualquer número inteiro possível da linguagem C. Utilizando o operador de resto, é possível delimitar o tamanho do número gerado. A segunda cria a semente de geração de aleatórios, sendo conveniente de ser usada com a função `time()`, que retorna o horário do computador, garantindo sementes sempre diferentes.

```
1 while (paintingFrameCheckOverlay(destiny, origin, height, widht))
2 {
3     height = (rand() % ((**destiny).height - (**origin).height + 1)
4 );
5     widht = (rand() % ((**destiny).widht - (**origin).widht + 1));
6 }
7 paintingFrameCopyPaint(destiny, origin, height, widht);
```

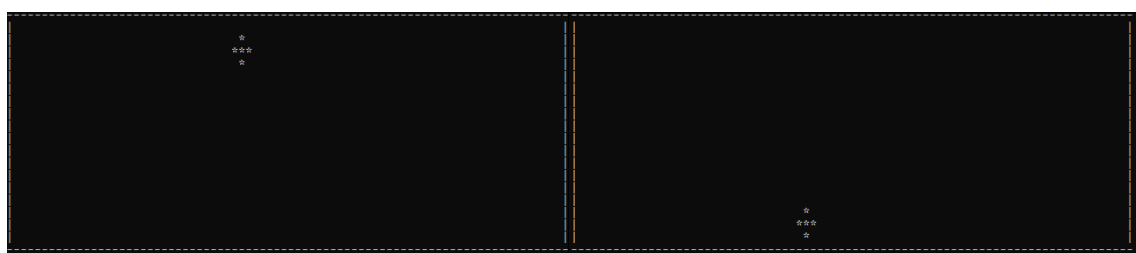


Figura 3. Mesmo desenho inserido em duas posições diferentes.

### 2.1.5. A operação surpresa

A operação surpresa gera imagens aleatórias do espaço, e tem um funcionamento bem simples, sendo uma paráfrase da função `paintingFrameInsertPaintInRandomPosition` (ver 2.1.4). Todos os desenhos usados dos planetas e estrelas estão na pasta `astro`. Algumas vezes a função apenas desenha alguns asteriscos que correspondem ao céu noturno visto por um observador distante, por isso é importante testar algumas vezes para ver as obras de arte mais complexas aparecerem.

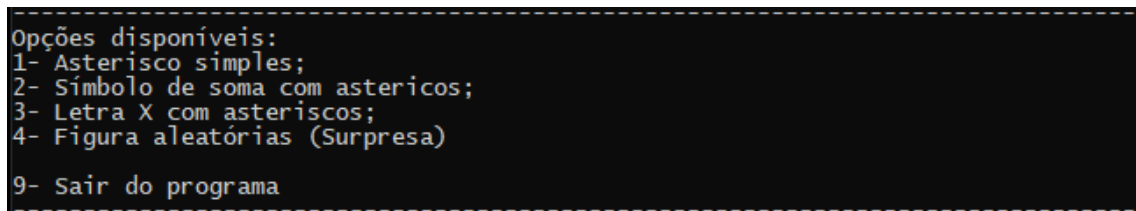
### 2.2. auxFunctions

Os arquivos `auxFunctions.c` e `auxFunctions.h` contêm definições auxiliares de operações, como mostrar o menu, exibir uma linha vertical etc.

```
1 bool auxFunctionsPrintHorizontalLine(unsigned short int width);
2 bool auxFunctionsPrintVerticalLine();
3 bool auxFunctionsPrintMenu();
4 bool auxFunctionsAskForType();
5 bool auxFunctionsAskForCopies();
```

### 2.3. O main.c

O arquivo principal é composto de `paintingFrames` tanto para o fundo do quadro a ser exibido, quanto para os desenhos básicos pedidos. Ele implementa um menu muito simples para chamar as operações do tipo abstrato de dados. Como a estrutura desenvolvida permite fácil alteração do tamanho, também foi adicionado a opção de trabalhar com desenhos maiores ou menores ao gosto do usuário.



```
Opções disponíveis:
1- Asterisco simples;
2- Símbolo de soma com asteriscos;
3- Letra X com asteriscos;
4- Figura aleatórias (Surpresa)
9- Sair do programa
```

Figura 4. Opções possíveis do menu implementado no `main.c`.

### 2.4. makefile

O `makefile` criado é extremamente simples, e permite utilizar tanto o compilador GCC quanto Clang. Este mostrou-se muito mais eficiente, pois é possível utilizar o flag `-O3` sem erros.

```
1
2 # Nome do executavel
3 TARGET=marombitte.o
4
5 # Compiladores
6 CC=gcc
7 CCO=clang
8
9 # Warnings
10 WARN=-Wall
```

```
11
12 # Main e .c dos TADS
13 SRC= main.c auxFunctions.c adt/paintingFrame.c
14
15 all:
16     $(CC) -o $(TARGET) $(SRC) $(WARN) -lm
```

### 3. Referências

Todas as imagens usadas em ASCII para a surpresa foram retiradas do site Asciiart - <https://www.asciart.eu/>

### 4. Conclusão

Através deste trabalho foi possível aprender um pouco mais sobre as famosas funções `rand()` e `srand()` da linguagem C, além de ter servido muito bem como aquecimento para a disciplina de Projeto e análise de algoritmos que virá. O maior desafio aqui foi projetar como os desenhos seriam representados no computador, e como a sobreposição poderia ser evitada. Utilizei para isso a solução mais simples que pensei. Também foi complicado escolher algo criativo para a opção quatro. Acabei optando por escolher algo que gosto desde criança: o espaço.