

Manual do Desenvolvedor: Canoagem

Elmadjian, Carlos Eduardo
elmadjian@linux.ime.usp.br

Morita, Ricardo Mikio
ricardom@linux.ime.usp.br

Santaella, Gil
gssantaella@gmail.com

8 de junho de 2013

Introdução



*"Nessa canoa furada
remando contra a maré
Não acredito em nada
até duvido da fé"*

Rita Lee

Este documento tem como papel descrever o funcionamento do executável de Canoagem para o desenvolvedor. Nas seções a seguir, descreveremos o funcionamento do programa.

1 Compilação

O programa principal está contido em `main.c`, o qual depende dos seguintes outros arquivos-fontes:

- **config.h** → Interface de `config.c`
- **config.c** → Contém rotinas de tratamento de entrada, de abertura de arquivos e de configuração do programa
- **debugger.h** → Interface de `debugger.c`
- **debugger.c** → Contém rotinas para o modo debugagem
- **grade.h** → Interface de `grade.c`
- **grade.c** → Contém rotinas de manipulação da grade geradora do rio
- **rio.h** → Interface de `config.c`
- **rio.c** → Contém rotinas de manipulação do tipo abstrato Rio
- **graficos.h** → Interface de `graficos.c` ***novo!***
- **graficos.c** → Funções relativas ao tratamento gráfico das entradas do jogo. ***novo!***

O arquivo `makefile` contém instruções de compilação para o programa `main` e suas dependências. Basta executar o `make` no mesmo diretório para gerar um executável do programa.

2 Implementação

2.1 main.c

Por padrão, o programa principal irá carregar um arquivo de configuração, o qual será utilizado para determinar as configurações iniciais do jogo. O programa também é responsável por chamar duas funções auxiliares:

- `menu()`: gera um menu inicial com três opções e é invocada quando o jogo não está e modo debug;
- `testeIntegridade()`: realiza testes automatizados e é invocada quando o parâmetro 2 é passado para o modo debug.

2.2 Módulo config

Este módulo é responsável pelo carregamento e processamento das entradas para o programa. Ele processa na seguinte ordem:

1. Leitura de um arquivo de entrada (ex. `config.txt`)
2. Leitura do `stdin`
3. Alterações no menu de configuração após inicialização

As duas primeiras etapas são executadas pela função `setEntradas()`. A última é executada pela função `mudaAtributos()`. Após o carregamento das variáveis, os outros módulos podem acessar os valores pelas funções `get` descrita a seguir:

Nome	Variável
<code>int getLeftMargin()</code>	Limite esquerdo do rio
<code>int getRightMargin()</code>	Limite direito do rio
<code>float getWaterSpeed()</code>	Velocidade máxima de cada casa da matriz
<code>int getSeed()</code>	Seed usado
<code>int getRefreshRate()</code>	Refresh Rate (em microssegundos)
<code>int getIsleDist()</code>	Distância mínima entre duas ilhas
<code>float getIsleProb()</code>	Probabilidade de aparecer uma ilha
<code>int getNumLines()</code>	Número de linhas da matriz
<code>int getNumColumns()</code>	Número de colunas da matriz
<code>int getNumIterations()</code>	Número de iterações até o fim do programa
<code>int getNumSeconds()</code>	Número de segundos até o fim do programa
<code>float getRiverFlux()</code>	Fluxo do rio em cada linha da matriz
<code>int getReportData()</code>	<code>rd=1</code> : Relatório; <code>rd=2</code> : teste de robustez
<code>int getDebugMode()</code>	Ignora conflitos na entrada, usado em <code>-rd2</code>
<code>char getWaterChar()</code>	Caractere que simboliza a água
<code>char getEarthChar()</code>	Caractere que simboliza a terra
<code>char getIsleChar()</code>	Caractere que simboliza a ilha

Por último, uma função que é usada em outros módulos é a `clearScreen()`, que limpa a tela. O restante das funções são de uso interno do módulo e são descritas brevemente aqui:

- `setParametro()`: Atribui um *int* à sua respectiva variável;
- `setParametrof()`: Atribui um *float* à sua respectiva variável;
- `setParametroc()`: Atribui um *char* à sua respectiva variável;
- `converteString()`: Soma cada caractere da string e retorna a soma;

- `checaAtributos()`: Verifica se há contradições nas entradas;
- `checaEntradas()`: Checa por erros e pede correções caso haja
- `queroInt()`: Pede um novo valor para a variável de tipo `int`;
- `queroFloat()`: Pede um novo valor para a variável de tipo `float`;
- `queroChar()`: Pede um novo valor para a variável de tipo `char`;
- `imprimeAtributos()`: Mostra o que tem em cada variável;
- `checaAtributosGraf()*novo*`: Retorna uma string de um erro encontrado no setup;

2.3 Módulo debugger

O módulo debugger apresenta um conjunto de rotinas com o intuito de testar três aspectos do programa: robustez, correção e variabilidade. Duas funções são indicadas para acompanhar o comportamento do programa “on the flow”:

- `printInfoLinha()`: mostra o fluxo calculado e a velocidade média em cada linha;
- `printInfoTopo()`: mostra o tamanho da grade e o tempo decorrido de geração de linhas do rio.

As demais funções são auxiliares:

- `calculaFluxo()`: usada por `printInfoLinha()`, calcula o fluxo numa linha a partir da soma das velocidades;
- `calculaVelMedia()`: também usada por `printInfoLinha()`, calcula a velocidade média de cada linha (considerando apenas os pontos com velocidade > 0);

E as seguintes rotinas são utilizadas em testes de correção e variabilidade:

- `calculaVarMargens()`: calcula variação de cada margem do rio;
- `calculaVarVelocidade()`: calcula variação das velocidades médias de cada linha;
- `verificaFluxo()`: verifica a correção do fluxo (uma vez que o fluxo deve ser constante em todo o rio);
- `printRelatorio()`: mostra um relatório final a partir do cálculo das funções de variação.

2.4 Módulo grade

O módulo grade apresenta funções de manipulação de matrizes (ou grades) do tipo abstrato Rio (conferir módulo rio).

Este módulo é composto por três funções:

- **printGrade()**: é encarregada de imprimir uma matriz inteira na tela. Portanto, é a função responsável por mostrar as atualizações feitas em cada linha por **geraRio()** (conferir o módulo rio), o que causa uma ilusão de movimento para o espectador;
- **alocaGrade()**: aloca espaço para uma matriz do tipo Rio. Essa função impede que não haja vazamento de memória no programa, uma vez que qualquer alteração na matriz será feita usando o espaço reservado apenas uma vez para a grade;
- **freeGrade()**: libera o espaço alocado por **alocaGrade()**.

2.5 Módulo rio

O módulo rio apresenta funções relacionadas à manipulação do tipo abstrato Rio (descrito em `rio.h`), uma *struct* que guarda um valor do tipo *char* (correspondente à representação gráfica do rio) e um valor do tipo *float* (correspondente à velocidade de um ponto na água).

A principal função deste módulo é a **geraRio()**, que coleciona chamadas a todas as demais funções do módulo por meio de **geraLinha()** e imprime na tela uma matriz do tipo Rio por um certo tempo, determinado ou não.

A função **geraLinha()** se ocupa de devolver uma linha para **geraRio()** de modo que:

- todos os pontos esteja preenchidos com terra ou água (**preencheTerreno()**);
- tenha sido gerada a probabilidade de incluir uma ilha ou não no rio (**geraIlha()**);
- tenha sido calculada as variações das margens direita e esquerda (**sorteiaMargens()**);
- cada ponto na água tenha uma velocidade (**preencheVelocidade()**);
- as velocidades tenham sido normalizadas (**normalizaVelocidade()**).

2.6 Módulo Gráfico **novo**

Este módulo apresenta funções pertinentes à parte visual do jogo. A parte gráfica é feita pela implementação de bibliotecas do Allegro 5.0, especificamente:

- `allegro.h`
- `allegro_image.h`
- `allegro_font.h`
- `allegro_ttf.h`
- `allegro_primitives.h`
- `allegro_image.h`

Este módulo é composto pelas funções abaixo:

- `criaJanela()`: Inicializa a janela do jogo;
- `destroiJanela()`: Destroi a janela usada pelo programa;
- `desenhaMenu()`: Renderiza o Menu inicial do jogo;
- `desenhaSetup()`: Renderiza o setup do jogo;
- `desenhaRio()`: Desenha o jogo;

3 Observações

Para saber mais detalhes técnicos sobre o funcionamento de cada rotina, por favor, confira as interfaces dos arquivos-fontes.

Para explicações mais detalhadas de implementação, veja os arquivos `.c` dos módulos correspondentes.