Documentação Jogo 20 perguntas

Gerado por Doxygen 1.8.11

Sumário

Capítulo 1

Índice dos Arquivos

1.1 Lista de Arquivos

Esta é a lista de todos os arquivos documentados e suas respectivas descrições:

Capítulo 2

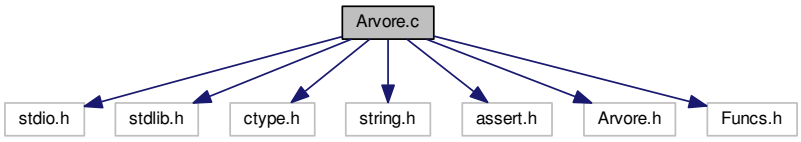
Arquivos

2.1 Referência do Arquivo Arvore.c

Arquivo que contem a biblioteca de manipulação e criação da arvore.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <assert.h>
#include "Arvore.h"
#include "Funcs.h"
```

Gráfico de dependência de inclusões para Arvore.c:



Definições e Macros

- #define _Primary_libraries

  Header de funções padrão, para I/O, manipulação de strings e asserts.

- #define _Arvore_library

  Header da biblioteca de arvore.

- #define _Funcs_library

  Header da biblioteca de funções (criação de arquivo e concatenação de strings).

Funções

- void Constroi_Manual (arvore **ainicio, char *no, unsigned int size)

  Função de criação da arvore de forma manual.
- void Constroi_TXT (arvore **ainicio, FILE *arq)

  Função de criação da arvore por .txt.
- void Salva_TXT (arvore **ainicio, FILE *arq)

  Função para salvar as perguntas da arvore em um .txt.
- void Le (arvore *arv)

  Função para a leitura da pergunta no nó da Arvore.
- void NavegaSim (arvore **pergunta)

  Função para a navegação e leitura da pergunta para o nó 'sim' da arvore.
- void NavegaNao (arvore **pergunta)

  Função para a navegação e leitura da pergunta para o nó 'nao' da arvore.
- void Desconstroi (arvore **ainicio)

  Função para apagar a arvore da memória.

2.1.1 Descrição Detalhada

Arquivo que contem a biblioteca de manipulação e criação da arvore.

Autor

Andre Garrido Damaceno

2.1.2 Definições e macros

2.1.2.1 #define _Primary_libraries

Header de funções padrão, para I/O, manipulação de strings e asserts.

Como esse arquivo contem a biblioteca da arvore, necessita dos headers padrões e de de funções auxiliares.

2.1.3 Funções

2.1.3.1 void Constroi_Manual (arvore ** *ainicio,* char * *no,* unsigned int *size*)

Função de criação da arvore de forma manual.

Parâmetros: Essa função recebe como parametro o endereço de um ponteiro de arvore 'arvore **ainicio' (para sua criação), uma string 'char *no' para a informação a respeito do nó atual para o usuário, um inteiro 'int size', para impedir a criação de mais que 20 níveis de perguntas (para garantir que o usuário poderá responder apenas 20 perguntas no máximo). A função não retorna nenhum parametro.

Tratamento de Erros: Caso haja erro de alocação de memória em 'Constroi_Manual' ou nas funções que ela chama, o programa é encerrado com erro.

Descrição: Essa função cria a arvore de acordo com as perguntas inseridas pelo usuário, sendo possível o usuário criar quantas perguntas quiser (com o limite de 1048576 perguntas), podendo parar de inserir quando quiser.

Assertivas de entrada: string 'no' não ser null, size ser no máximo 20.

Requisitos: A função deve criar uma árvore binária com todas as perguntas inseridas pelo usuario, mostrando para o usuário sempre, o lugar da árvore em que ele está. Deve também inserir apenas no máximo 20 níveis de perguntas.

Hipóteses: A árvore tem o tamanho alocado ideal, a string tem seu tamanho alocado ideal, a arvore deve possuir apenas 20 níveis de perguntas. Ao final, toda memória alocada para a string deve ser liberada, apenas a memoria alocada da arvore deve se manter.

Assertivas de saida: alocações internas para strings desalocadas.

Interface explicita: estrutura de árvore '∗∗ainicio', string '∗no', profundidade da árvore 'size'

Interface implicita: Não há.

Contrato na especificação: A função deve receber uma árvore vazia ou incompleta, uma string com a localização do nó atual e o tamanho atual da profundidade da árvore. A saída de cada chamada vai ser ou um nó null ou um nó de árvore com uma pergunta, dependendo apenas se o usuário quis ou não inserir algo naquele nó, e se ainda não havia sido preenchidas os 20 níveis da árvore.

2.1.3.2 void Constroi_TXT (arvore ∗∗ *ainicio,* FILE ∗ *arq*)

Função de criação da arvore por .txt.

Parametros: Essa função recebe como parametro o endereço de um ponteiro de arvore 'arvore ∗∗ainicio' (para sua criação) e um arquivo (para a leitura das perguntas e criação da arvore). A função não retorna nenhum parametro.

Tratamento de erros: Caso haja erro de alocação de memória em 'Constroi_TXT' o programa é encerrado com erro.

Descrição: Essa função cria a arvore de acordo com um arquivo de texto aberto pelo usuário, sendo os nós nulos identificados por pontos '.'.

Assertivas de entrada: a função deve receber um arquivo não nulo.

Requisitos: A função deve criar uma árvore binária de acordo com o arquivo de txt passado como parâmetro.

Hipoteses: O tamanho da árvore é alocado com o tamanho correto, o arquivo txt é lido de forma correta. Ao final, deve ser alocada uma árvore em memória idêntica à árvore encontrada no arquivo txt.

Assertivas de saida: Não há, pois o arquivo pode criar a árvore, ou ele ser em branco, não criando a árvore.

Interface explicita: estrutura de árvore '∗∗ainicio', arquivo '∗arq'

Interface implicita: Manipulação de arquivos pela variável '∗arq'

Contrato na especificação: A função deve receber uma árvore vazia ou incompleta, um arquivo de texto não nulo. A função deve criar ou aumentar os nós da árvore de acordo com o arquivo txt (sendo possível inclusive não criar nenhum nó ou árvore, caso o arquivo txt esteja em branco).

2.1.3.3 void Desconstroi (arvore ** *ainicio*)

Função para apagar a arvore da memória.

Parametros: Essa função recebe como parametro o endereço do ponteiro da arvore 'arvore **ainicio' e não retorna nenhum parametro.

Tratamento de erros: Caso seja inserido um nó invalido não NULL, pode ser que ocorra um erro de SegFault.

Descrição: A função navega recursivamente para o ultimo nó sim, em seguida o ultimo nó nao, e vai apagando a arvore. A função checa se o nó é NULL, para evitar erros e conseguir apagar de forma recursiva.

Assertivas de entrada: Nó da árvore válido (não NULL)

Requisitos: A função deve receber uma árvore não nula, e apagar todos seus nós, liberando toda a memória alocada da árvore.

Hipoteses: A funçaõ verifica adequadamente os nós da árvore, sempre libera a memória de forma adequada.

Assertivas de saida: Nó da árvore ser NULL.

Interface explicita: estrutura de árvore '**ainicio'

Interface implicita: Não há.

Contrato na especificação: A função deve receber um endereço de árvore alocado na memória, e deve então desalocar toda a memória alocada pela árvore, liberando todos os nós.

2.1.3.4 void Le (arvore * *arv*)

Função para a leitura da pergunta no nó da Arvore.

Parametros: Essa função recebe como parametro um ponteiro de arvore 'arvore *a1', não retorna nenhum parametro.

Tratamento de erros: Caso o ponteiro não seja valido (NULL), a função não faz nada (não havendo erros).

Descrição: A função apenas checa se o ponteiro é valido, caso seja, imprime na tela a pergunta.

Assertivas de entrada: Ponteiro da árvore válido (não NULL).

Requisitos: A função deve receber uma árvore não nula e imprimir a pergunta que se encontra no nó.

Hipoteses: A função verifica se a árvore é ou não válida e imprime apenas as perguntas de nós validos.

Assertivas de saida: não há.

Interface explicita: estrutura de árvore '**arv'

Interface implicita: Não há.

Contrato na especificação: A função recebe uma árvore como parametro, verifica se ela é valida, apenas caso ela seja válida a função imprime a pergunta do nó. Caso não seja valida nada acontece.

2.1.3.5 void NavegaNao (arvore ** *pergunta*)

Função para a navegação e leitura da pergunta para o nó 'nao' da arvore.

Parametros: Essa função recebe como parametro o endereço do ponteiro da arvore, e não retorna nenhum parametro.

Tratamento de erros: Caso em algum ponto o nó seja NULL, a função não faz nada, apenas retorna.

Descrição: A função checa se o nó atual é valido e se o nó 'nao' é valido, caso sejam, o apontador passa a apontar para o nó 'nao' e a pergunta é lida.

Assertivas de entrada: A árvore recebida como parametro e o nó que será navegado não podem ser NULL.

Requisitos: A função deve navegar para o nó 'nao' da arvore recebida como parametro, em seguida ler a pergunta desse nó.

Hipoteses: A função lê apenas perguntas validas com nós validos.

Assertivas de saida: não há, pois o nó pode .

Interface explicita: estrutura de árvore '**pergunta'

Interface implicita: Não há.

Contrato na especificação: A função deve receber um nó de árvore, avaliar se ele é valido (não NULL), caso seja válido verificar o nó 'nao', caso o nó 'nao' seja válido, navega para ele e imprime a pergunta do nó.

2.1.3.6 void NavegaSim (arvore ** *pergunta*)

Função para a navegação e leitura da pergunta para o nó 'sim' da arvore.

Parametros: Essa função recebe como parametro o endereço do ponteiro da arvore, e não retorna nenhum parametro.

Tratamento de erros: Caso em algum ponto o nó seja NULL, a função não faz nada, apenas retorna.

Descrição: A função checa se o nó atual é valido e se o nó 'sim' é valido, caso sejam, o apontador passa a apontar para o nó 'sim' e a pergunta é lida.

Assertivas de entrada: A árvore recebida como parametro e o nó que será navegado não podem ser NULL.

Requisitos: A função deve navegar para o nó 'sim' da arvore recebida como parametro, em seguida ler a pergunta desse nó.

Hipoteses: A função lê apenas perguntas validas com nós validos.

Assertivas de saida: não há, pois o nó pode .

Interface explicita: estrutura de árvore '**pergunta'

Interface implicita: Não há.

Contrato na especificação: A função deve receber um nó de árvore, avaliar se ele é valido (não NULL), caso seja válido verificar o nó 'sim', caso o nó 'sim' seja válido, navega para ele e imprime a pergunta do nó.

2.1.3.7 void Salva_TXT (arvore ∗∗ *ainicio,* FILE ∗ *arq*)

Função para salvar as perguntas da arvore em um .txt.

Parametros: Essa função recebe como parametro o endereço do ponteiro de uma arvore 'arvore ∗∗ainicio', e o arquivo de texto a ser salvo as perguntas da arvore 'FILE ∗arq'. A função não retorna nenhum parametro.

Tratamento de erros: Caso a função receba um arquivo null, nada ocorre.

Descrição: A função salva a pergunta no arquivo .txt, caso o nó seja NULL, é salvo um '.' no arquivo. Caso o arquivo seja inexistente, a função não faz nada. Caso a arvore seja inexistente, a função salva apenas um '.' no .txt.

Assertivas de entrada: A função deve receber um arquivo não nulo.

Requisitos: Irá ser passado por parâmetro uma árvore e o arquivo txt a ser salvo, a função irá então salvar os dados da arvore no arquivo txt.

Hipoteses: A função checa se o arquivo existe, se a arvore ja chegou no fim e salva todas as perguntas da arvore de forma adequada.

Assertivas de saida: não há.

Interface explicita: estrutura de árvore '∗∗ainicio', arquivo '∗arq'

Interface implicita: Manipulação de arquivo pela variável '∗arq'

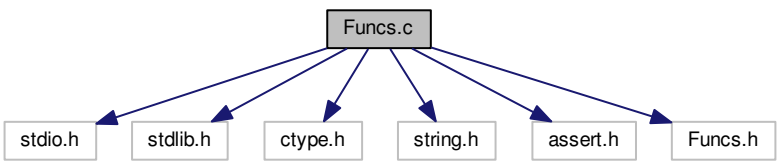
Contrato na especificação: A função irá salvar a árvore por completo no arquivo txt recebido como parametro, inclusive salvando no txt as partes da árvore que são NULL, representando-os com um '.'.

2.2 Referência do Arquivo Funcs.c

Arquivo que contem a função de respostas do usuário, concatenação de string e de criar arquivos txt (abrir ou salvar).

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <assert.h>
#include "Funcs.h"
```

Gráfico de dependência de inclusões para Funcs.c:



Definições e Macros

- #define _Primary_libraries

  Header de funções padrão, para I/O, manipulação de strings.
- #define _Funcs_library

  Header da biblioteca de funções (criação de arquivo e concatenação de strings).

Funções

- unsigned int Resposta (unsigned int tipo)

  Função para pegar o input especifico de opções do usuario.
- FILE ∗ CriaArquivo (char ∗type, char ∗opcao)

  Função de Criação de arquivos (abertura ou leitura).
- char ∗ PosicaoNo (char ∗no, char ∗filho)

  Função de concatenação de string.

2.2.1 Descrição Detalhada

Arquivo que contem a função de respostas do usuário, concatenação de string e de criar arquivos txt (abrir ou salvar).

Autor

Andre Garrido Damaceno

2.2.2 Definições e macros

2.2.2.1 #define _Primary_libraries

Header de funções padrão, para I/O, manipulação de strings.

Como esse arquivo contem apenas funções auxiliares, necessita apenas dos headers padrões.

2.2.3 Funções

2.2.3.1 FILE∗ CriaArquivo (char ∗ *type,* char ∗ *opcao*)

Função de Criação de arquivos (abertura ou leitura).

Parametros: Essa função recebe como parametro o tipo de abertura 'char ∗type' ("r" - para read, "w" - para write) e também uma string 'char ∗opcao' para ser impresso na tela (informando ao usuário se o arquivo esta sendo aberto ou salvo). A função retorna o arquivo 'FILE ∗'.

Tratamento de erros: Caso o arquivo não exista, ou ele será criado (no caso do tipo "w" - write), ou a função retornará NULL no 'FILE ∗'.

Descrição: Essa função apenas abre um arquivo e o retorna para o usuário.

Assertivas de entrada: string 'type' e 'opcao' não NULLs.

Requisitos: A função deve perguntar ao usuário qual o nome do arquivo e abrir ou criar um arquivo de acordo com o parâmetro 'type'.

Hipoteses: A função deve abrir o arquivo de forma adequada de acordo com a forma de abertura 'type' requisitado, retornando o arquivos aberto/criado.

Assertivas de saida: Arquivo aberto/criado não ser NUL

Interface explicita: string com a forma de abertura do arquivo '∗type' e string com a opção de abertura '∗opcao'

Interface implicita: Manipulação de arquivo pela variável '∗arq'.

Contrato na especificação: A função deve receber a forma de abertura/criação do arquivo 'type', e uma string com a informação para o usuário do que está ocorrendo. Então, deve ser aberto/criado o arquivo de acordo com o parâmetro 'type' e o nome informado pelo usuário, e por fim, deve ser retornado o arquivo.

2.2.3.2 char∗ PosicaoNo (char ∗ *no,* char ∗ *filho*)

Função de concatenação de string.

Parametros: Essa função recebe como parametro duas strings 'char ∗no' e 'char ∗filho', concatena a string 'filho' na string 'no', e retorna essa string 'char ∗'.

Tratamento de erros: Caso o computador negue a alocação de memória, o programa é finalizado.

Descrição: Inicialmente a função aloca memória suficiente para a concatenação das strings, em seguida copia a string 'no' para a memoria alocada, por fim, concatena a string 'filho' na memoria alocada, retornando assim a string com 'no' e 'filho' concatenados devidamente.

Assertivas de entrada: As strings 'no' e 'filho' não podem ser NULLs.

Requisitos: A função deve concatenar as duas strings recebidas 'no' e 'filho' e retornar a string concatenada.

Hipoteses: A função aloca devidamente memória suficiente para a string final concatenada, e faz a concatenação de forma correta, concatenando no sentido 'no' e 'filho'.

Assertivas de saida: A string com o resultado da concatenação não deve ser NULL, O tamanho da string retornada deve ser o tamanho de 'no' + 'filho', e o final da string deve conter o caracter '\0'.

Interface explicita: primeira string para concatenação '∗no', segunda string para concatenação '∗filho'

Interface implicita: Não há.

Contrato na especificação: A função deve receber duas strings 'no' e 'filho' não nulas, deve então alocar uma nova string 'noFilho' que deve possuir o tamanho de 'no' e 'filho' juntas, em seguida colocar o resultado da concatenação de 'no' e 'filho' em 'noFilho' e retornar 'noFilho'.

2.2.3.3 unsigned int Resposta (unsigned int *tipo*)

Função para pegar o input especifico de opções do usuario.

Parametros: Essa função recebe como parametro um inteiro 'int tipo', que especifica o tipo de opção que o usuário terá e retorna um inteiro que representa a opção selecionada (escrita) pelo usuario.

Tratamento de erros: Caso o usuario tenha respondido algo invalido, é mencionada as respostas que a pergunta espera, e dada a chance do usuario responder novamente, caso contrario, é retornado o equivalente da resposta do usuario pelo inteiro. Caso haja um erro de leitura pelo 'scanf' (usuario digita mais que 6 caracteres), apenas é mencionada a mensagem dos tipos da resposta disponivel multiplas vezes, qualquer outro tipo de erro é desconhecido o comportamento (pois estariam dependendo das funções 'strcmp', 'strlen' e toupper).

Descrição: Essa função inicialmente lê a resposta escrita pelo usuário e delimita as respostas para o Tipo da pergunta, sendo 'simples' - para perguntas de 'sim' ou 'nao', 'multipla' - para perguntas de 'sim', 'nao', 'editar' ou 'apagar', e 'inicializacao' - para perguntas de 'abrir' ou 'criar'.

Assertivas de entrada: O valor de entrada deve estar entre 0 (pergunta multipla) e 2 (pergunta de inicialização).

Requisitos: A função deve receber a resposta do usuário de acordo com o contexto (se é uma pergunta multipla, de inicialização ou simples), e retornar uma resposta válida apenas. A função deve ficar perguntando ao usuário até que uma resposta válida seja respondida.

Hipoteses: A leitura da opção é feita de forma correta, de acordo com o tamanho do vetor que fica armazenada a resposta do usuário. O loop é feito apenas quando há uma resposta incorreta, saindo dele sempre que a resposta correta ser respondida.

Assertivas de saida: O retorno tem que estar entre 0 (Rsim) e 5 (Rabrir).

Interface explicita: número que identifica o tipo da pergunta 'tipo'

Interface implicita: Não há.

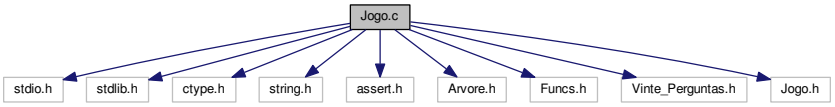
Contrato na especificação: A função recebe como parametro o tipo da pergunta e deve retornar com a resposta do usuário de acordo com o tipo da pergunta recebida. Caso ela receba um tipo, e é respondido uma resposta válida em outros tipos mas não no tipo recebido, a resposta deve ser considerada inválida.

2.3 Referência do Arquivo Jogo.c

Arquivo que contem a Main e uma função de chamadas para executar o jogo.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <assert.h>
#include "Arvore.h"
#include "Funcs.h"
#include "Vinte_Perguntas.h"
#include "Jogo.h"
```

Gráfico de dependência de inclusões para Jogo.c:



Definições e Macros

- #define _Primary_libraries

  Header de funções padrão, para I/O, manipulação de strings.
- #define _Arvore_library

  Header da biblioteca de arvore.
- #define _Funcs_library

  Header da biblioteca de funções (criação de arquivo e concatenação de strings).
- #define _Vinte_Perguntas_library

  Header da biblioteca de estruturação (execução) do jogo de 20 perguntas.
- #define _Jogo_library

  Header da biblioteca que inicializa o jogo.

Funções

- int main ()

  Main de inicialização do jogo.
- void Jogo_init (void)

  Função de inicialização do jogo.

2.3.1 Descrição Detalhada

Arquivo que contem a Main e uma função de chamadas para executar o jogo.

Autor

Andre Garrido Damaceno

2.3.2 Definições e macros

2.3.2.1 #define _Primary_libraries

Header de funções padrão, para I/O, manipulação de strings.

Como esse arquivo contem a função que inicializa o jogo, são necessários todos arquivos headers que o programa utiliza.

2.3.3 Funções

2.3.3.1 void Jogo_init (void)

Função de inicialização do jogo.

Parametros: A função não recebe nem retorna nenhum parâmentro.

Tratamento de erros: Caso haja algum erro em alocação de memória, o programa é encerrado. Outros casos variam, caso a arvore seja null, eventualmente havera a opção de tentar recriar a arvore. Caso haja algum outro erro totalmente imprevisto, ou as funções internas o conterão, ou o programa finalizará com erro.

Descrição: Nessa função, primeiramente é populado os dados na arvore (por arquivo de texto ou por criação manual). Em seguida, inicializa-se o jogo ao chamar a função 'Vinte_Perguntas()', ao fim, é perguntado ao usuário se quer salvar os dados de seu jogo em um arquivo .txt, e por fim é finalizada a execução (deletando a arvore da memoria).

Assertivas de entrada: árvore ser NULL, arquivo ser NULL.

Requisitos: A função deve chamar a função de popular os dados da árvore (de acordo com a escolha do usuario, por arquivo texto ou manualmente), em seguida chamar a função de executar o jogo, por fim, verificar se o usuario quer salvar a árvore do Jogo em um arquivo de texto, e desalocar a memoria da árvore.

Hipoteses: Toda memória alocada é feita com o tamanho correto, toda memória alocada é desalocada após o uso, todos os arquivos de texto abertos são fechados.

Assertivas de saida: A árvore tem que ser NULL (desalocada).

Interface explicita: Não há.

Interface implicita: Manipulação de arquivo pela variável '∗arq'.

Contrato na especificação: A função não recebe nenhum parametro nem retorna nenhum parametro, deve apenas chamar as funções principais capazes de fazer o jogo rodar, ou seja, em ordem, deve ser populada a árvore com perguntas, em seguida ser chamada a função de execução do jogo e por fim deve ser dada a escolha de salvar a arvore do jogo em um arquivo de texto. Devem ser finalizadas todas as memorias alocadas e arquivos abertos.

2.3.3.2 int main ( )

Main de inicialização do jogo.

Parametros: Não recebe nenhum parâmetro de entrada, retorna um inteiro no fim da execução, sendo 0 - execução bem sucedida, outros valores caso contrário.

Tratamento de erros: Não há, apenas há chamadas de funções. Os tratamentos são feitos dentro das funções chamadas.

Descrição: A main apenas chama a função de inicialização 'Jogo_init()', que chama as funções do jogo em ordem lógica para sua execução normal.

Assertivas de entrada: Não há

Requisitos: Chamar a função de inicialização 'Jogo_init'.

Hipoteses: A função é chamada com sucesso e o jogo é inicializado.

Assertivas de saida: Não há

Interface explicita: Não há.

Interface implicita: Não há.

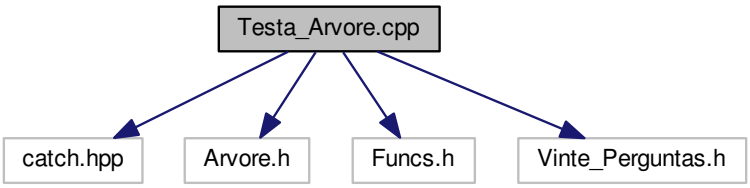
Contrato na especificação: A função deve apenas chamar a função 'Jogo_init'.

2.4 Referência do Arquivo Testa_Arvore.cpp

Arquivo que contem os testes do jogo de 20 perguntas.

```
#include "catch.hpp"
#include "Arvore.h"
#include "Funcs.h"
#include "Vinte_Perguntas.h"
```

Gráfico de dependência de inclusões para Testa_Arvore.cpp:



Definições e Macros

- #define CATCH_CONFIG_MAIN
- #define _Catch

  Header da biblioteca de testes.
- #define _Arvore_library

  Header da biblioteca de arvore.
- #define _Funcs_library

  Header da biblioteca de funções (criação de arquivo e concatenação de strings).
- #define _Vinte_Perguntas_library

  Header da biblioteca de estruturação (execução) do jogo de 20 perguntas.

Funções

- TEST_CASE ("Creating a tree from user input","Prove that the tree is created")

  Teste da função 'Constroi_Manual'.
- TEST_CASE ("Creating a tree from a file","Prove that the tree is created")

  Testes da função 'Constroi_TXT' - Criação normal da arvore.
- TEST_CASE ("Trying to create a tree from an non existing file","Prove that the tree is not created")

  Testes da função 'Constroi_TXT' - tentativa de criar arvore por arquivo Null.
- TEST_CASE ("Trying to navigate to save a NULL tree to file","Prove that the txt saves '.'")

  Testes da função 'Salva_TXT' - tentativa de salvar arvore NULL.
- TEST_CASE ("Saving a tree to file","Prove that the txt saves the tree")

  Testes da função 'Salva_TXT' - tentativa de salvar arvore existente.
- TEST_CASE ("Saving tree to NULL file","Prove that the function does nothing and contains the program")

  Testes da função 'Salva_TXT' - tentativa de salvar arquivo inexistente.
- TEST_CASE ("Freeing an existing tree","the tree is freed")

  Teste da função 'Desconstroi' - Apagando uma arvore existente.

- TEST_CASE ("Freeing a NULL tree","the program is contained")

  Teste da função 'Desconstroi' - Apagando uma arvore inexistente.
- TEST_CASE ("Reading a tree question","tree is unmodified and question is read")

  Teste da função Le - lendo arvore existente.
- TEST_CASE ("Trying to read NULL tree","Program is contained and function does nothing")

  Teste da função Le - lendo arvore inexistente.
- TEST_CASE ("Trying to navigate to '->sim' and '->nao' and reading the question","Tree goes to specific navigation and reads the question")

  Testes de navegação (sim e nao) - arvore existente.
- TEST_CASE ("Trying to navigate to '->sim' and '->nao' whith NULL tree","Program is contained and function returns 2")

  Testes de navegação (sim e nao) - arvore inexistente.
- TEST_CASE ("Creating/Opening a file (read) and (write)","Function opens/creates the file")

  Teste da função 'CriaArquivo' - arquivos existentes "r" e "w" e arquivos inexistentes.
- TEST_CASE ("Function that concatenates strings","Should concatenate the string")

  Testando a função 'PosicaoNo' - Concatenação de strings existente e inexistente.
- TEST_CASE ("Testing user's answers in game","Get only 'sim', 'nao', 'editar', 'apagar' answer and nothing else")

  Testando a função 'Resposta' - teste para o tipo 'simples', 'multipla' e inicializacao.
- TEST_CASE ("Receiving null tree or object was not guessed by Vinte_Perguntas","The function should be contained, should ask the user to add more questions if less than 20 answers")

  Teste da função 'Vinte_Perguntas' - Arvore inexistente.
- TEST_CASE ("Receiving 20 questions and testing 'apagar' and 'editar' in Vinte_Perguntas","The function should ask the questions and navigate in the tree, and execute its functions 'apagar', 'editar'")

  Teste da função 'Vinte_Perguntas' - Arvore existente, cheia, edição, apagar.
- TEST_CASE ("Creating childs at the end in 'sim' and 'nao'","Should normaly create childs")

  Teste da função 'Pergunta_Final'.

2.4.1 Descrição Detalhada

Arquivo que contem os testes do jogo de 20 perguntas.

Autor

Andre Garrido Damaceno

2.4.2 Definições e macros

2.4.2.1 #define CATCH_CONFIG_MAIN

Como esse arquivo contem os testes, necessita dos headers de toda a biblioteca do jogo.

2.4.3 Funções

2.4.3.1 TEST_CASE ("Creating a tree from user input" , "Prove that the tree is created")

Teste da função 'Constroi_Manual'.

Testes feitos e criterio de aceitação: Teste de criação de uma arvore NULL - criterio de aceitação é a arvore ser null e o programa se conter. Teste de criação no nó pai apenas - criterio de aceitação nó pai criado e os nós filhos NULL. Teste de criação no pai e no filho 'SIM' - criterio de aceitação pai nao ser NULL, filho 'sim' nao ser NULL, filho 'sim' 'sim' ser NULL, filho 'sim' 'nao' ser NULL e filho 'nao' ser NULL. Teste de criação no pai e no filho 'NAO' - criterio de aceitação pai nao ser NULL, filho 'nao' nao ser NULL, filho 'nao' 'sim' ser NULL, filho 'nao' 'nao' ser NULL e filho 'sim' ser NULL. Teste de criação de um nó pai, um nó filho 'sim', um nó filho 'nao' - criterio de aceitação pai nao ser NULL, filho 'nao' nao ser NULL, filho 'nao' 'sim' ser NULL, filho 'nao' 'nao' ser NULL, filho 'sim' não ser NULL filho 'sim' nao ser NULL e filho 'sim' 'sim' ser NULL. Todos os testes foram bem sucedidos.

2.4.3.2 TEST_CASE ("Creating a tree from a file" , "Prove that the tree is created")

Testes da função 'Constroi_TXT' - Criação normal da arvore.

Testes feitos e criterio de aceitação: Teste de criação da arvore com um arquivo existente (onde os nós pai, filho 'sim' filho 'nao' existem na arvore), os criterios de aceitação foram os nós descritos serem iguais à frases escritas no teste (comparação feita por strcmp), e os dos filhos 'sim' e 'nao' serem NULL. Todos os testes foram bem sucedidos.

2.4.3.3 TEST_CASE ("Trying to create a tree from an non existing file" , "Prove that the tree is not created")

Testes da função 'Constroi_TXT' - tentativa de criar arvore por arquivo Null.

Foi feito um teste, abrindo um arquivo inexistente pela função 'fopen', e passado o arquivo para a função 'Constroi↩ _TXT', os criterios de aceitação são a Arvore ser NULL, e o programa não ter problemas em sua execução. O teste passou com sucesso, e tudo ocorreu como esperado.

2.4.3.4 TEST_CASE ("Trying to navigate to save a NULL tree to file" , "Prove that the txt saves '.'")

Testes da função 'Salva_TXT' - tentativa de salvar arvore NULL.

Foi feito um teste, abrindo um arquivo inexistente na forma "w", então foi salva uma arvore NULL no arquivo. Em seguida, foi aberto esse arquivo e construida a arvore a partir dele com a função 'Constroi_TXT', e por fim, o criterio de aceitação é a arvore criada ser NULL. O teste passou com sucesso, tudo ocorreu como esperado.

2.4.3.5 TEST_CASE ("Saving a tree to file" , "Prove that the txt saves the tree")

Testes da função 'Salva_TXT' - tentativa de salvar arvore existente.

Foi aberto um arquivo txt existente com dados de arvore, criada a arvore pela função 'Constroi_TXT', em seguida foi salva a arvore criada pela função 'Salva_TXT' em um outro arquivo de texto inexistente ('Perguntas2.txt'), por fim, para verificar se tudo ocorreu como esperado, foi aberto esse arquivo txt, feito a arvore novamente, e checada todas as perguntas que existiam no arquivo, e também se todos os nós inexistentes da arvore eram NULL. Todos os criterios passaram, tudo ocorreu como esperado.

2.4.3.6 TEST_CASE ("Saving tree to NULL file" , "Prove that the function does nothing and contains the program")

Testes da função 'Salva_TXT' - tentativa de salvar arquivo inexistente.

Foi aberto um arquivo txt na função "r", em seguida foi tentado salvar uma arvore NULL em um arquivo inexistente, os criterios de aceitação são a função se conter, a arvore ser NULL, o arquivo ser NULL. Todos os testes passaram com sucesso.

2.4.3.7 TEST_CASE ("Freeing an existing tree" , "the tree is freed")

Teste da função 'Desconstroi' - Apagando uma arvore existente.

Foi criada uma arvore por um txt existente, em seguida, foi verificado que a arvore não era NULL, por fim, foi chamada a função de desconstrução, e o criterio de aceitação é que a função apagasse a arvore e seu ponteiro fosse Null no final. Todos os testes passaram com sucesso.

2.4.3.8 TEST_CASE ("Freeing a NULL tree" , "the program is contained")

Teste da função 'Desconstroi' - Apagando uma arvore inexistente.

Foi passada para a função 'Desconstroi()' uma arvore NULL, o resultado esperado e criterio de aceitação é que o programa se contenha e que a arvore continue sendo NULL. Todos os testes passaram com sucesso.

2.4.3.9 TEST_CASE ("Reading a tree question" , "tree is unmodified and question is read")

Teste da função Le - lendo arvore existente.

Foi criada uma arvore por um arquivo txt existente e passado o ponteiro da arvore para a função 'Le()', o criterio de aceitação é que a mensagem da pergunta apareça na tela e o ponteiro da arvore não seja alterado. Todos os testes passaram com sucesso.

2.4.3.10 TEST_CASE ("Trying to read NULL tree" , "Program is contained and function does nothing")

Teste da função Le - lendo arvore inexistente.

Foi passado um ponteiro NULL de arvore para a função 'Le()', o criterio de aceitação é que função não fizesse nada e se contenha e o ponteiro da arvore não fosse alterado de NULL. Todos os testes passaram com sucesso.

2.4.3.11 TEST_CASE ("Trying to navigate to '->sim' and '->nao' and reading the question" , "Tree goes to specific navigation and reads the question")

Testes de navegação (sim e nao) - arvore existente.

Foi criada uma arvore por um arquivo txt, e inicializado as variaveis navegasim, naveganao e ainiciobackup com o endereço da arvore e verificado se todas continham o endereço da arvore, em seguida, foi feita a navegação NavegaSim em navegasim e NavegaNao em naveganao e o criterio de aceitação foi se o resultado dos ponteiros eram iguais aos ponteiros da arvore original no nó 'sim' e 'nao', por fim, mais uma vez foi chamada as funções NavegaSim e NavegaNao e verificado se os nós eram NULL (para checar se eram iguais à arvore original que é NULL). Todos os testes e verificações foram bem sucedidas.

2.4.3.12 TEST_CASE ("Trying to navigate to '->sim' and '->nao' whith NULL tree" , "Program is contained and function returns 2")

Testes de navegação (sim e nao) - arvore inexistente.

Para esse teste, foi criado um ponteiro NULL, e feito a navegação de NavegaSim e NavegaNao passando como parametro o ponteiro NULL, para criterio de aceitação, verifica-se se a função se conteve e se os ponteiros continuam sendo NULL. Todos os testes e verificações foram bem sucedidos.

2.4.3.13 TEST_CASE ("Creating/Opening a file (read) and (write)" , "Function opens/creates the file")

Teste da função 'CriaArquivo' - arquivos existentes "r" e "w" e arquivos inexistentes.

Primeiramente é aberto um arquivo com a função "r", e criado uma arvore a partir desse arquivo, o teste de aceitação é que todos os nós do arquivo sejam iguais à strings colocadas no teste. Em seguida, cria-se um arquivo com a função "w", e é feito o teste de escrita, usando a função 'Salva_TXT' para salvar a arvore no arquivo de texto, o criterio de aceitação é a função se conter e o arquivo ser criado com sucesso. Por fim, checa-se a abertura de um arquivo inexistente (passando o nome de um arquivo que nao se encontra no computador), o criterio de aceitação é que o arquivo seja NULL. Todos os testes e criterios passaram com sucesso.

2.4.3.14 TEST_CASE ("Function that concatenates strings" , "Should concatenate the string")

Testando a função 'PosicaoNo' - Concatenação de strings existente e inexistente.

São declaradas strings e feito criterios de aceitação, usando a função 'strcmp' para comparar as strings concatenadas com strings digitadas no teste, testando inclusive a concatenação de duas strings vazias, sendo o resultado valido da concatenação string vazia. Todos os testes e criterios passaram com sucesso.

2.4.3.15 TEST_CASE ("Testing user's answers in game" , "Get only 'sim' , 'nao' , 'editar' , 'apagar'answer and nothing else")

Testando a função 'Resposta' - teste para o tipo 'simples', 'multipla' e inicializacao.

Os testes são feitos ao passar todas as possiveis combinações (maiusculo ou minusculo) dos resultados das possiveis respostas, e feito uma comparação se a resposta é como a esperada. O criterio de aceitação é que todas as variaveis em todas as suas combinações possiveis de maiusculo e minusculo sejam reconhecidos, e que as mensagens de erro ao digitar um caracter invalido seja exibido de forma correta na tela. Todos os testes e requisitos passaram com sucesso.

2.4.3.16 TEST_CASE ("Receiving null tree or object was not guessed by **Vinte_Perguntas**" , "The function should be *contained,* should ask the user to add more questions if less than 20 answers")

Teste da função 'Vinte_Perguntas' - Arvore inexistente.

Nesse teste, foi testado o comportamento da função 'Vinte_Perguntas' quando recebe uma arvore Null, sendo o primeiro requisito que a arvore continuasse NULL, e que no segundo requisito que a arvore fosse criada (no minimo um nó). Todos os testes e requisitos passaram com sucesso.

2.4.3.17 TEST_CASE ("Receiving 20 questions and testing 'apagar' and 'editar' in **Vinte_Perguntas**" , "The function should ask the questions and navigate in the *tree,* and execute its functions 'apagar' , 'editar'")

Teste da função 'Vinte_Perguntas' - Arvore existente, cheia, edição, apagar.

Foi testado inicialmente o comportamento da função 'Vinte_Perguntas' ao receber e navegar até o final de uma arvore com as 20 perguntas preenchidas, o criterio de aceitação é que o jogo finalizasse se o usuario chegou ou não no objeto que ele estava pensando. O outro teste foi de edição do nó principal, o criterio de aceitação foi que a string editada seja igual à string digitada no teste, e por fim, foi testado diversas situações de apagar, sendo as situações de apagar o nó pai, apagar os nós filhos e desistir de apagar um nó, para cada situação, respectivamente o criterio de aceitação é que o nó pai seja NULL, nós filhos sejam NULLs, nada ocorrece ao desistir de apagar e o jogo retorne normalmente na ultima pergunta não respondida (que foi tentada ser apagada). Todos os testes e requisitos passaram com sucesso.

2.4.3.18 TEST_CASE ("Creating childs at the end in 'sim' and 'nao'" , "Should normaly create childs")

Teste da função 'Pergunta_Final'.

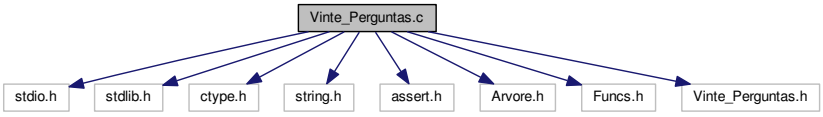
Foi aberto um arquivo txt e criado uma arvore, em seguida passada essa arvore para a função 'Pergunta_Final', com o intuido da criação de um novo nó 'sim' na arvore, o criterio de aceitação é que o nó fosse criado com sucesso e que seus filhos fossem NULL, em seguida, novamente foi feito o mesmo teste só que para o nó 'nao', o criterio é que o nó 'nao' seja existente e seus filhos sejam NULL. Todos os testes passaram com sucesso.

2.5 Referência do Arquivo Vinte_Perguntas.c

Arquivo que contem a biblioteca de estruturação (execução) do jogo de 20 perguntas.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <assert.h>
#include "Arvore.h"
#include "Funcs.h"
#include "Vinte_Perguntas.h"
```

Gráfico de dependência de inclusões para Vinte_Perguntas.c:



Definições e Macros

- #define _Primary_libraries

  Header de funções padrão, para I/O, manipulação de strings.
- #define _Arvore_library

  Header da biblioteca de arvore.
- #define _Funcs_library

  Header da biblioteca de funções (criação de arquivo e concatenação de strings).
- #define _Vinte_Perguntas_library

  Header da biblioteca de estruturação (execução) do jogo de 20 perguntas.

Funções

- void Vinte_Perguntas (arvore **anavega, unsigned int numero_respostas)

  Função de execução do programa.
- void Pergunta_Final (arvore **anterior, arvore **ainicio, unsigned int numero_respostas, unsigned int opcao)

  Função que verifica se o computador acertou o objeto, e cria mais perguntas caso não tenha acertado (se o usuário quiser e ainda não tenham sido respondidas 20 perguntas).

2.5.1 Descrição Detalhada

Arquivo que contem a biblioteca de estruturação (execução) do jogo de 20 perguntas.

Autor

Andre Garrido Damaceno

2.5.2 Definições e macros

2.5.2.1 #define _Primary_libraries

Header de funções padrão, para I/O, manipulação de strings.

Como esse arquivo contem a biblioteca de execução, necessita dos headers padrões, de funções auxiliares, e de estruturação do jogo.

2.5.3 Funções

2.5.3.1 void Pergunta_Final (arvore ∗∗ *anterior,* arvore ∗∗ *ainicio,* unsigned int *numero_respostas,* unsigned int *opcao*)

Função que verifica se o computador acertou o objeto, e cria mais perguntas caso não tenha acertado (se o usuário quiser e ainda não tenham sido respondidas 20 perguntas).

Parametros: Essa função recebe como parametro o endereço do ponteiro da arvore 'arvore ∗∗anterior', o endereço do apontador do inicio da arvore 'arvore ∗∗ainicio', o numero de perguntas ja respondidas 'int numero_respostas' e a última opção selecionada 'int opcao'. A função não retorna nenhum parametro.

Tratamento de erros: Os casos de erro são os mesmos das funções 'Vinte_Perguntas()' e 'Constroi_Manual()', pois depende dessas funções e de alocação de memoria do computador, sendo assim, a execução é terminada caso haja algum erro de alocação.

Descrição: Essa função tem o objetivo de checar se o computador conseguiu chegar na resposta do objeto que o usuário pensava. Dessa forma, a função dá a opção de criar novas perguntas para alcançar esse objetivo (caso o usuário tenha respondido menos que 20 perguntas ou um nó foi apagado) e por fim, retorna ao jogo (ou mostra o resultado caso ja tenham sido respondidas as 20 perguntas).

Assertivas de entrada: numero_respostas deve ser menor que 21, apontador da árvore 'anterior' não pode ser NULL.

Requisitos: A função deve verificar se o usuário achou a resposta e se ja foram respondidas as 20 perguntas. Caso ainda não tenha chegado na resposta e ja tenha sido respondidas 20 perguntas, nada ocorre. Caso não tenha chegado na resposta e ainda não tenham sido respondidas as 20 perguntas, o usuário tem a opção de inserir mais perguntas até que as 20 perguntas sejam completadas, por fim o jogo retorna. Caso já tenha achado a resposta do objeto pensado pelo usuário, o programa finaliza.

Hipoteses: Todas as memórias alocadas são feitas com o tamanho e a forma adequada, as navegações com os ponteiros são feitas de forma correta e com cuidado para não se perder as referências.

Assertivas de saida: Não há.

Interface explicita: estrutura de árvore '∗∗ainicio' e '∗∗anterior', quantidade de respostas 'numero_respostas', ultima opção selecionada pelo usuário 'opcao'.

Interface implicita: Não há.

Contrato na especificação: A função deve verificar se o usuário já chegou na resposta desejada, caso ja tenha, uma mensagem é exibida. Caso ainda não tenha chegado, verifica-se se já foram respondidas 20 perguntas. Caso ja tenha sido respondido, é exibida uma mensagem de consolo. Caso contrário, é disponibilizada uma opção de inserir mais perguntas para se chegar na resposta (com o limite de 20 perguntas no total). No final, o jogo retorna na útlima pergunta respondida (que não tinha a resposta).

2.5.3.2 void Vinte_Perguntas (arvore ** *anavega,* unsigned int *numero_respostas*)

Função de execução do programa.

Parametros: Essa função recebe como parametro o endereço do ponteiro da arvore 'arvore **anavega' (para as possiveis mudanças na arvore como apagar, editar, navegar e criar novos nós) e um inteiro 'int numero_respostas', para saber quantas perguntas ja foram respondidas pelo usuario. Essa função não retorna nenhum parametro.

Tratamento de erros: Como essa função de execução utiliza a maior parte de todas as funções criadas, os erros estão relacionados a essas funções. Mas todas as funções inclusive essa, foi desenvolvida para conter erros e finalizar o programa apenas se um erro de alocação de memoria ocorrer.

Descrição: A função inicialmente alerta ao usuário que o jogo irá começar e analisa se a arvore é vazia ou o numero_respostas é menor que 19 (ja foram respondidas 20 perguntas). Em seguida é lida a pergunta para o usuário e ele pode navegar pelas perguntas (respondendo 'sim' ou 'nao') ou editar/apagar uma pergunta. No fim, é perguntado ao usuário se seu objeto pensado foi descoberto. Caso tenha sido, o jogo é finalizado, caso não tenha sido e o usuário ainda não tenha respondido 20 perguntas, é perguntado se o usuário deseja inserir mais perguntas para o programa descobrir o objeto. Ao fim das inserções, o jogo recomeça a partir do ponto da ultima pergunta respondida pelo usuário.

numero_respostas menor que 21

Requisitos: A função deve percorrer as perguntas da árvore de acordo com as respostas do usuário, também ter a capacidade de verificar se ja foram respondidas 20 perguntas. Ao fim, também oferecer a opção de inserir mais perguntas caso ainda não tenham sido respondidas as 20, e a resposta não tenha sido encontrada.

Hipoteses: As alocações de memória são feitas com o tamanho ideal e de forma correta, as navegações com os ponteiros são feitas de forma adequada.

Assertivas de saida: Não há.

Interface explicita: estrutura de árvore '**anavega', quantidade de respostas 'numero_respostas'

Interface implicita: Não há.

Contrato na especificação: A função recebe a árvore com as perguntas e a quantidade de respostas já respondidas, ela deve então navegar na árvore de acordo com as respostas do usuário, até o fim da árvore (onde pode ser que tenha ou não a resposta do objeto pensado pelo usuário). Caso não chegue na resposta e não tenham sido respondidas 20 perguntas ainda, a função dá a opção de inserir mais perguntas até que a resposta seja alcançada ou as 20 perguntas sejam criadas.

Índice Remissivo