DOCUMENTAÇÃO TRABALHO PRÁTICO 3

João Igor De Andrade Oliveira - 2018072409

Vinícius De Castro Mendes Morais - 2018020867

FASE DE ESPECIFICAÇÃO

O projeto foi dividido em diversas funções, com o intuito de deixar mais organizado o ambiente e mais prático de serem feitas quaisquer alterações. Foi também utilizada uma biblioteca encontrada na internet, cujo nome é AndreRenaud/PDFGen (fonte: https://github.com/AndreRenaud/PDFGen). Essa biblioteca foi o que possibilitou a criação de uma saída em formato PDF, conforme desejado para o calendário. Todas as funções criadas serão explicadas na fase projeto. A TAD utilizada no projeto foi uma árvore, já que ela possui a característica de ter um custo baixo tanto para busca quanto para inserção e remoção. Dessa forma, a utilização desta permite uma melhor execução do projeto, com mais velocidade e melhor performance. Ela é baseada em nós, de forma que cada nó possui um valor/informação. Assim, para execução correta do programa, foi necessário incluir a pdfgen.h e calccalendario.h, sendo que essa primeira proveu da biblioteca baixada enquanto a outra foi criado pela dupla. A execução do programa é ocorre tendo por base as preferências do usuário. Um menu com as opções vai surgindo à medida que são feitas as escolhas.

PROJETO

Será feita uma explicação sobre cada função utilizada e criada, separadamente.

- Funções da biblioteca adicionada

•struct pdf_info info

Essa função é responsável por definir as informações do PDF, como título, autor, data e assunto.

•pdf_set_font

Essa função é responsável pela definição de qual fonte será usada no que for escrito no PDF, como por exemplo Times New Roman, Arial.

•int pdf_add_text(struct pdf_doc *pdf, struct pdf_object *page, const char *text, int size, int xoff, int yoff, uint32_t colour);

Essa função é responsável por adicionar o texto no PDF. O primeiro elemento é o PDF em si, logo em seguida a página do PDF desejada, depois o texto a ser escrito, seguido de tamanho, posição em relação ao eixo x, posição em relação ao eixo y e a cor desejada.

•int pdf_add_jpeg(struct pdf_doc *pdf, struct pdf_object *page, int x, int y, int display_width, int display_height, const char *jpeg_file);

Essa função se assemelha a do texto, porém é para adicionar uma imagem ao PDF. Além dessa diferença, ela pede para inserir um valor para a largura, para a altura e o nome/diretório do arquivo da imagem.

-Funções criadas

•char* calcmes(int *mes);

Essa função é para quando o usuário inserir um mês em número, converter esse mês para uma palavra, ou seja, o nome do mês. Essa função foi utilizada dentro de uma outra para escrever o texto no PDF, já explicada acima. O mês 13 retorna janeiro e 0 retorna dezembro devido ao cálculo que será feito durante a execução do programa, usado para o calendário do mês anterior e do mês seguinte. Quando é impresso o calendário menor, existe a possibilidade de troca de ano entre o mês anterior e o mês que se deseja fazer a consulta ou então entre o mês seguinte e o que se deseja fazer a consulta. Portanto, como somamos 1 a variável do mês para averiguar o mês seguinte, se o mês “principal” for 12, teremos o mês 13, sendo ele janeiro, seguinte a dezembro. Ocorre o semelhante pro mês anterior, se o mês “principal” for 1, ao subtrair 1 teremos o mês 0, referente a dezembro.

•char* calcmesING(int *mes);

Essa função é a mesma da anterior, porém traduzida pro inglês, caso seja vontade do usuário utilizar este idioma.

•int language();

Essa função foi criada para disponibilizar ao usuário o poder de escolher se quer que o programa seja executado em inglês ou português. Ao inserir 0, o programa executa em português e, ao inserir 1, executa em inglês.

•int escolher_ano();

Pede ao usuário para escolher o ano do calendário a ser impresso.

•int escolher_mes();

Pede ao usuário para escolher o mês do calendário a ser impresso.

•int choose_year();

Pede ao usuário para escolher o ano do calendário a ser impresso em inglês.

•int choose_month();

Pede ao usuário para escolher o mês do calendário a ser impresso em inglês.

•int escolhe_mini_calendario();

Pede ao usuário que escolha como deseja a impressão do calendário, com o calendário principal e 2 menores com os meses seguinte e anterior, ou calendário principal com o mês seguinte.

•choose_mini_calendar();

Pede ao usuário que escolha como deseja a impressão do calendário, seja com o calendário principal e 2 menores com os meses seguinte e anterior, ou calendário principal com o mês seguinte, em inglês.

•int calcula_calendario(int *dia, int *mes, int ano, int hh);

Essa função já é um pouco mais complexa. Os argumentos dela são o número de dias do mês, o mês, o ano e uma variável auxiliar hh. Foi feito para cada mês um if, de forma que cada mês tenha o número correto de dias. Dentro de cada if existe uma variável auxiliar que é usada para o cálculo, a aux3. Ela divide em grupos os meses que começam em dias da semana iguais. Por exemplo, os meses de janeiro e outubro sempre começam no mesmo dia da semana (menos quando é ano bissexto) e, portanto, possuem o mesmo valor de aux. Isso faz com que não seja necessário fazer os cálculos para os 12 meses, apenas para a quantidade de aux3 diferentes. Há também a auxiliar aux2, cuja função é alterar os dias que cada mês começa. Quando forem anos bissextos, ela tem seu valor mantido, quando não for, ela é reduzida de um, para que no final, o somatório dos auxiliares mude e, por consequência, o dia que o mês se inicia. Esse processo ocorre no mês de fevereiro, já que ele que altera todo o restante. Outra auxiliar é a aux1, que é utilizada para cálculo do dia da semana que começa o mês. Inicialmente, o valor dela é o número do ano subtraído de 1900, já que não serão utilizados anos anteriores a ele. Foi utilizada dentro das funções para saber se o ano é bissexto, o que altera a quantidade de dias do mês de fevereiro e os meses seguintes devem começar um dia após do que iria começar caso fosse um ano comum. Ao calcular o resto da divisão do aux1 por 4, pode-se determinar se é bissexto e fazer as devidas alterações. Caso o valor retornado seja 0, significa que o ano é bissexto, caso contrário, não é. Após todas a iterações, o valor de aux1 é alterada para a soma de todas auxiliares (aux1 + aux2 + aux3). Dessa forma, o valor de aux1 é referente ao dia da semana em que termina o mês, sendo 0 referente ao domingo e 6 ao sábado. Assim, a função se encerra retornando o valor final de aux1, que será usada posteriormente para a impressão do calendário.

•void imprime_calendario(int aux1, int *dia, int *mes, int ano, struct pdf_doc *pdf, Arvore *raiz);

Essa função é responsável pela impressão do calendário em versão PDF. Ela utiliza do valor da aux1, da quantidade de dias que o mês possui, o mês, o ano e a arvore binária que foi criada (será explicada a frente). Ao analisar a pixelização do PDF, foi possível determinar os valores das posições x e y dos números referentes aos dias da semana. É utilizada uma aux3 que é referente ao número da semana. Uma aux2 é usada apenas para a conversão do valor int i para char aux2, em um sprintf, já que a função de impressão de texto no PDF aceita apenas char. Então, o processo é feito começando do último dia do mês e vai até o primeiro dia do mês. Assim, seguindo os valores da aux1, referentes ao dia da semana, os números são colocados em sua devida posição em x e, seguindo os valores da aux3, os números são colocados em sua devida posição em y. Um comando for é utilizado para que seja executado o processo até que se complete o número total de dias do mês.

•Arvore* cria_arvore();

É a função responsável por alocar memória suficiente para armazenar todos os eventos do arquivo txt e para criar a árvore.

•void compara_dia(int i, int *mes, int ano, Arvore *raiz, struct pdf_doc *pdf);

A função recursiva é utilizada para a análise dos dias de acordo com o desejado pelo usuário. Ela fará uma busca pelo dia, mês e ano do arquivo txt e irá imprimir no calendário todos os eventos do mês no calendário, em seus respectivos dias. Além disso, dentro dela foram colocadas as funções que, ao analisar o tipo do evento, um ícone referente a esse tipo será adicionado ao lado do evento no calendário. Essa função compara_dia analisa os elementos da árvore, ponto a ponto, separados tanto pela esquerda e pela direita (conforme deve ser feito em uma árvore) até que seja encontrado na árvore a raiz que aponta para o ano de mesmo valor que o ano inserido pelo usuário. O mesmo ocorre para o dia e para mês. Após atingir os 3 valores, é impresso no calendário os eventos referentes a esta data.

•Arvore* ler_evento();

Foi criada para que os eventos dentro do arquivo sejam lidos e separados em uma arvore. Dentro dela há o uso de uma função responsável por encadear os eventos em uma árvore, a encadear_evento (que será explicada abaixo). Então, ela aloca o espaço de memória necessário pra árvore, usando a função cria_arvore, e logo após lê os eventos do arquivo. Assim, ao ler o evento, ele é adicionado à arvore, usando a função encader_evento. Dessa forma, a leitura e o encadeamento dos eventos ocorrem simultaneamente, sem a necessidade de se separar em outras funções, o que poderia gerar uma complexidade maior para o programa. A função para quando todos os eventos já tiverem sido lidos e encadeados.

•int encadear_evento(char Tipo, int Dia, int Mes, int Ano, char frase[], Arvore *raiz);

Essa é a função responsável por separar todos os eventos, de forma organizada e correta, dentro de uma árvore. Ela utiliza if e else para fazer as comparações dos elementos dela. Assim, ela separa os eventos primeiramente analisando o ano em que eles estão, depois o mês e, por seguinte, por dia. Eles são divididos no formato da árvore, de forma que são organizados separadamente por ano, dentro de cada ano por mês e dentro de cada mês por dia. Assim, os eventos do dia são colocados após o dia. Também são separados por cada tipo de evento, de forma que fiquem organizados dentro dos dias por A (aniversário), O (outros) e V (viagem).

•void imprime_arvore(Arvore *raiz);

Essa função é utilizada para imprimir no terminal após a busca pelo tipo de evento, dia ou mês, o que o usuário escolher no menu.

•void imprime_minidir(int aux1m, int *dia, int *proxMes, int proxAno, struct pdf_doc *pdf);

Essa função se assemelha a imprime_calendario, tem a mesma ideia, porém, é a impressão do calendário menor na parte superior direita do PDF, referente ao mês seguinte.

•void imprime_miniesq(int aux1mm, int *dia, int *MesAnt, int AnoAnt, struct pdf_doc *pdf);

Essa função se assemelha a imprime_calendario, tem a mesma ideia, porém, é a impressão do calendário menor na parte superior esquerda do PDF, referente ao mês anterior.

•void busca(Arvore *raiz) // void search(Arvore *raiz)

Essas funções são responsáveis por executarem a busca dos eventos que o usuário escolher no menu. Eles são buscados por tipo de evento ou então pela data. Após a escolha do tipo de busca, pede para serem inseridos a data ou então qual o tipo do evento. Após isso, aparece no próprio terminal de execução o resultado da busca.

•int encadear_dia(char Tipo, int Dia, int Mes, int Ano, char frase[], Arvore *raiz)

Essa função é responsável por fazer o encadeamento dos eventos, de forma semelhante à função encadear_evento. Entretanto, a outra não ordenava por dia, enquanto essa já possui tal diferença. Assim, sua utilização é necessária quando o usuário faz a escolha da busca por mês e ano no terminal, para que imprima os eventos em ordem crescente da data.

ANÁLISE DE COMPLEXIDADE

Devido à utilização de uma árvore de busca, a complexidade do programa tornou-se reduzida e dependente do número de eventos inseridos no arquivo. Portanto, no melhor caso, ao custo para execução do algoritmo é O(log N) e no pior caso, O(N), sendo N o número de eventos a serem lidos no arquivo. Para as impressões do calendário, há uma complexidade constante, haja vista que não depende do número de eventos, e sim dos dias de cada mês, que são fixos. Para fazer a impressão de todos os eventos é utilizada a função da árvore, ou seja, é impresso o direto do algoritmo de imprimir a árvore, o que leva a uma complexidade igual a da árvore. Portanto, no pior caso, o algoritmo completo tem custo O(N), enquanto no caso médio e melhor, O(logN).