Relatório T1 INE5408

Aluno: Jan Bortolanza - 20203632

Dentro do código existem identificadores para cada explicação dos blocos. Foi utilizada a implementação de pilha e fila feita em aula.

ID 1:

Para o problema de validação do XML primeiramente foi utilizada a biblioteca fstream para leitura do arquivo. O conteúdo desse arquivo é lido e salvo em uma string utilizando a função getline() dentro de um loop while. Essa função booleana lê uma linha do arquivo e salva na variável linha, retornando False quando não existem mais linhas para serem lidas. Cada linha é concatenada à variável xml, que por fim é passada para a função validarXML.

ID 2:

Explicação geral da função validarXML:
Para a verificação eu utilizei um loop while que percorre todos os elementos da string passada. Cada caractere é avaliado, se não houver nenhuma condição de erro a função retorna True.

ID 2.2:

Se o caractere avaliado for uma abertura de tag (<) o código utiliza a função find de strings que recebe como argumento uma posição e um char. A partir desse local definido por pos ela verifica se existe uma ocorrência do char(>) passado, se não houver é retornado ‘npos’, significando que uma tag foi aberta e não foi fechada, deixando o XML como inválido.
Caso contrário a posição do fechamento será salva e utilizada no método substr para extrair o identificador. Esse método gera uma nova string de uma str original entre os dois pontos passados.
Se o primeiro caractere do identificador for ‘/’ significa o fechamento de uma marcação. Caso a pilha esteja vazia significa que existe um fechamento sem abertura, portanto XML inválido, e se o topo da pilha não conter o identificador que está sendo fechado a pilha não está aninhada corretamente.
Se o primeiro caractere não for ‘/’ é feito o push do identificador na pilha e passa para a próxima posição.
Após todo o XML ser avaliado o único caso de erro restante é quando tags foram abertas e não fechadas. O trecho final verifica se a pilha está vazia (significando que o aninhamento está correto) e retorna True para main.

ID 3:

A função processaXML é responsável por dividir a string xml que foi validada em N cenários para serem processados pela rotina principal, armazenando-os em uma fila, garantindo que a ordem correta de processamento seja mantida.
Essa função não utiliza nenhum método novo em comparação à validaXML. Ela procura pela string “<cenario>” e salva a sua posição ao encontrar, após isso procura por “</cenario>” e gera uma substring com o conteúdo entre essas duas tags. Essas strings são colocadas em uma fila e o loop busca pelo próximo cenário.

ID 3.1:

A função getData é uma cópia de processaXML que recebe como parâmetro as tags de abertura e fechamento do que se deseja extrair do cenário e retorna um inteiro. Foi criada para extrair os valores de X e Y, assim como largura e altura da matriz.

ID 3.2:

É uma versão de getData que retorna uma string ao invés de int. Utilizada para extrair o nome do cenário e uma string contendo o conteúdo da matriz.

ID 4:

Essa é a função principal do código, ela recebe uma string com um cenário individual e retorna uma string com o resultado a ser impresso no terminal.
Inicialmente a rotina utiliza as funções getData para extrair os dados necessários para criação das matrizes e posição inicial do robô.
A biblioteca “vector” de C++ foi utilizada para criação das matrizes pois ela faz a alocação dinâmica de memória automática, necessário para que o XML 6 possa ser computado, pois haverão problemas com estouro de memória se a alocação das matrizes for feita estaticamente. O preenchimento das matrizes é bem simples, dois loops for que percorrem cada elemento e atribui um valor a ele.

O algoritmo implementado para o robô foi o que foi passado na descrição do projeto, utilizando uma fila de pares de coordenadas. O passo inicial desse algoritmo é atribuir 1 à posição inicial do robô e passar para a fila. Um loop while foi utilizado para passar por todos os itens dessa estrutura FIFO.
Inicialmente uma coordenada nova é buscada e armazenada, esse é o ponto onde o robô se encontra na matriz. Após isso é projetado o movimento da máquina nas quatro direções possíveis utilizando um loop for com as variáveis DX e DY, essas coordenadas são armazenadas e passam por um check para decidir se é uma posição válida de movimento para o robô e , consequentemente, deve ser limpa. O condicional if testa primeiramente se a posição atual existe na matriz, se na matriz E ela tem valor 1 e se na matriz R tem valor 0 (garantindo que o robô não fique travado limpando a mesma posição), caso tudo seja verdadeiro essa coordenada é colocada na fila e seu valor em R passa a ser 1.
A contagem da área a ser limpa é similar ao preenchimento das matrizes, só é contado o número de uns em R.

Nesse algoritmo existe um caso de erro que acontece quando a posição inicial do robô é 0. Como no começo é atribuído 1 à essa posição ela se torna válida, por mais que em E ela não seja. Isso faz com que no caso do robô começar cercado por zeros o resultado ainda seja 1 e, em alguns casos (como o P1030144), seja criada uma “ponte” para uma região maior válida que o robô não deveria conseguir acessar. Felizmente para consertar esse erro só é necessário checar se E[x][y] é 0 antes de executar a rotina.
Terminado tudo rotina retorna à main a string que deve ser impressa contendo nome e área a ser limpa.

Conclusões :

Acredito que 95% das dificuldades que tive na implementação do projeto foram causadas pela minha inexperiência com C++. Passei a maior parte do tempo pesquisando na documentação de string e aprendendo a instanciar as variáveis e estruturas de dados utilizadas. Depois que percebi que o algoritmo do robô era sobre preenchimento a sua implementação foi bem simples pois já tinha feito um código similar em POO 2.
Como referência utilizei o site https://cplusplus.com/ para documentação da linguagem e o https://stackoverflow.com/ para verificar exemplos de como dividir o XML de acordo com as tags.

Figuras:

As figuras mostram os passos que o robô percorreria na matriz:

Passo 1:
Robo = {4,2}
{3,2} e {4,3} são adicionados à fila

1	0	1	1	0
1	1	1	1	1
1	1	1	1	1
0	1	1	1	0
0	0	0	0	0

Passo 2:
Robo = {3,2}
{2,2} {3,1} {3,3} sao adicionados e {3,2} = 0