Trabalho Prático 1 - Identificação de Objetos Oclusos

1 Descrição do problema

A empresa Jolambs deseja melhor o desempenho dos seus jogos através de melhorias no seu algoritmo de apresentação de objetos na tela. Uma estratégia de melhoria é explorar a oclusão de objetos para reduzir a complexidade das cenas, ou seja, as cenas vão conter menos objetos por apresentar apenas os objetos (ou partes deles) visíveis.
A Figura 1 apresenta um exemplo em 1 dimensão contendo 6 objetos, onde cada objeto é representado por um segmento de reta paralela ao eixo X. Os objetos são apresentados na parte superior da figura, enquanto a "cena´´ resultante é apresentada na parte inferior da figura.
O programa a ser projetado e desenvolvido recebe como entrada um arquivo que pode conter três tipos de linha:

Objeto A linha do tipo objeto se inicia com `O` e contem o identificador do objeto, a sua posição inicial (que é o centro do objeto) e a largura do mesmo.

O seu formato é `O <objeto> <x> <y> <largura>`.

Movimento: A linha do tipo movimento se inicia com `M` e contém uma marca de tempo (um inteiro positivo), o identificador do objeto e a nova posição dele.

O seu formato é `M <tempo> <objeto> <x> <y>`.

Cena: A linha do tipo cena se inicia com `C` e contem apenas uma marca de tempo.

O seu formato é `C <tempo>`.

Um exemplo de entrada válida é mostrada na Figura 2. Note que os objetos são declarados antes de qualquer movimentação.
Já a saída apresenta, para cada linha, um objeto visualizado, total ou parcialmente. A linha, iniciada por S, contem uma marca de tempo, identificador de objeto, e a posição inicial e final de sua visualização naquele momento. O seu formato é `S <tempo> <objeto> <inicio> <fim>`. Por exemplo, considerando a entrada acima, a saída no tempo 10 é mostrada na Figura 3. Esta saída é representada graficamente na parte inferior da Figura 1.
Para entendermos melhor a tarefa a ser realizada, vamos discutir o problema de construção de cenas. Um algoritmo simples de construção de cenas descartando objetos oclusos é mostrado na Figura 4.
Você deve implementar um programa que gera cenas de objetos unidimensionais, como descrito. Para implementar o programa, você deve projetar, implementar e validar dois tipos abstratos de dados:

Objeto: O TAD objeto armazena os objetos que podem aparecer numa cena. Além da criação dos objetos, lembre-se que você tem que criar funções que atualizam a posição dos objetos e também suportam a geração da cena.

Cena: O TAD cena armazena os objetos ou partes deles que vão aparecer numa cena gerada no momento indicado. Lembre-se que você tem que criar funções que colocam objetos (ou parte deles) numa cena e gravar a cena (colocar num arquivo de saída).

Faz parte do trabalho a definição dos TADs e suas funcionalidades. O seu trabalho também consiste em melhorar os algoritmos utilizados atualmente, em particular investigar o compromisso entre manter o vetor de objetos ordenado a cada alteração ou ordenar apenas quando a geração de cena é invocada.

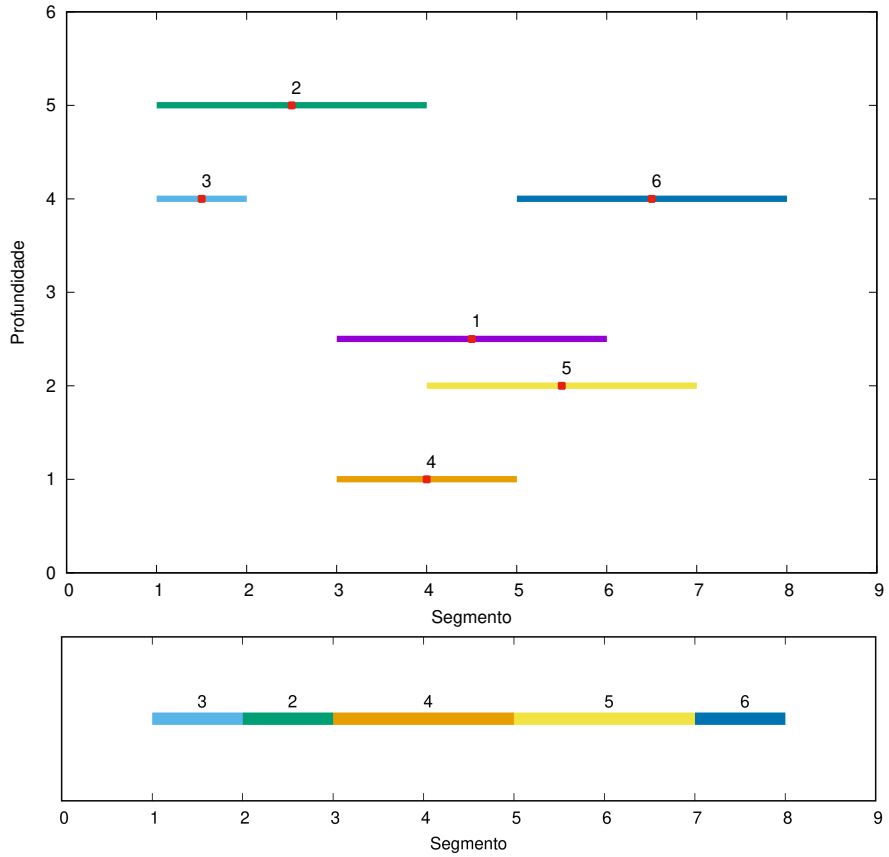

Figura 1: Exemplo de execução do algoritmo de identificação de oclusão. Acima podemos ver a entrada, que contem 6 segmentos. Abaixo podemos ver o resultado do algoritmo, onde apenas 5 dos segmentos são mostrados, alguns parcialmente (o segmento 1 está ocluso).

```
O 1 4 2.5 3
O 2 2.5 5 3
O 3 2 3 1
O 4 4 2 2
O 5 6 3 3
O 6 5 4 3
M 5 2 2.5 5
M 6 1 4.5 2.5
M 6 3 1.5 4
M 7 6 6.5 4
M 7 5 5.5 2
M 7 4 4 1
C 10
```

Figura 2: Exemplo de entrada

```
S 10 2 2 3
S 10 3 1 2
S 10 4 3 5
S 10 5 5 7
S 10 6 7 8
```

Figura 3: Exemplo de saída

```
void geraCena(objeto_t * vobj, int numobj,
              cena_t * cena, int numcena){
    OrdenaPorY(vobj, numobj);
    for (int i=0; i<numobj; i++){
        if (objetoVisivel(vobj[i])){
            adicionaObj(vobj[i],cena,&numcena);
        }
    }
}
```

Figura 4: Algoritmo para construção de cenas

2 Análise Experimental

A análise experimental tem por objetivo avaliar o compromisso entre a frequência de ordenação e de geração de cenas. Para avaliar esse compromisso, sugerimos os seguintes passos:

1. Defina métricas de desorganização do vetor de objetos. Por exemplo, sempre que atualizar um objeto, verifique se ele ainda está ordenado em relação aos seus vizinhos. Se não estiver mais ordenado, registre em um contador.

2. Defina limiares de desorganização do vetor. Esses limiares vão definir quando o vetor vai ser reordenado. Um limiar nulo significa que o vetor sempre vai ser reordenado quando algum objeto for atualizado. Já um limiar infinito significa que o vetor vai ser ordenado apenas quando a cena for gerada.

3. Projete o seu experimento, variando o número de objetos (quanto mais objetos, maior o custo de construir a cena), a largura dos objetos (quanto maiores os objetos, menos complexa a cena, pois haverá mais oclusões), a densidade espacial dos objetos (quanto mais denso o espaço de objetos, menos complexa a cena, pois haverá mais oclusões), a sua velocidade (quanto mais rápido os objetos se movem, mais o vetor se desorganiza e maior a necessidade de ordenação), entre outros.

4. Execute os seus experimentos e avalie os resultados, indicando os parâmetros escolhidos para cada configuração de experimento.

Análise experimental tem que ser cuidadosa para que as medidas sejam válidas. Para ilustrar alguns desses cuidados, suponha que você tem que realizar uma análise utilizando um vetor de estruturas, por exemplo objetos, onde cada estrutura possui uma chave (identificador de objeto) e um conteúdo, no caso, atributos do objeto.

Um primeiro ponto a ressaltar é que tanto a chave quanto o conteúdo não podem ser alocados dinamicamente, sob pena de não serem contíguos em termos de estrutura, afetando a localidade de referência e o perfil de acesso à memória. Desta forma, é necessário declarar estaticamente essa estrutura.

3 Pontos Extra

Os pontos extra tem por objetivo aumentar a flexibilidade e eficiência do TAD, como por exemplo:

- Estenda o seu TAD para lidar com objetos que não sejam mais paralelos ao eixo X.
- Estenda o seu TAD para lidar com objetos que sejam 2D ou 3D.

4 Como será feita a entrega

4.1 Submissão

A entrega do TP1 compreende duas submissões:

VPL TP1: Submissão do código a ser submetido até **03/10/2025, 23:59** (o sistema vai ficar aberto madrugada adentro, mais para evitar problemas transientes de infraestrutura). Detalhes sobre a submissão do código são apresentados na Seção 4.3.

Relatório TP1: Arquivo PDF contendo a documentação do TP, assim como a avaliação experimental, conforme instruções, a ser submetido até **03/10/2025, 23:59** (o sistema vai ficar aberto madrugada adentro, mais para evitar problemas transientes de infraestrutura). Detalhes sobre a submissão de relatório são apresentados na Seção 4.2.

4.2 Documentação

A documentação do trabalho deve ser entregue em formato **PDF** e também **DEVE** seguir o modelo de relatório que será postado no `minha.ufmg`. Além disso, a documentação deve conter **TODOS** os itens descritos a seguir **NA ORDEM** em que são apresentados:

1. **Capa:** Título, nome, e matrícula.
2. **Introdução:** Contém a apresentação do contexto, problema, e qual solução será empregada.
3. **Método:** Descrição da implementação, detalhando as estruturas de dados, tipos abstratos de dados (ou classes) e funções (ou métodos) implementados.
4. **Análise de Complexidade:** Contém a análise da complexidade de tempo e espaço dos procedimentos implementados, formalizada pela notação assintótica.
5. **Estratégias de Robustez:** Contém a descrição, justificativa e implementação dos mecanismos de programação defensiva e tolerância a falhas implementados.
6. **Análise Experimental:** Apresenta os experimentos realizados em termos de desempenho computacional[1], assim como as análises dos resultados.

[1] Para este trabalho não é necessário analisar a localidade de referência.

7. **Conclusões:** A Conclusão deve conter uma frase inicial sobre o que foi feito no trabalho. Posteriormente deve-se sumarizar o que foi aprendido.

8. **Bibliografia:** Contém fontes utilizadas para realização do trabalho. A citação deve estar em formato científico apropriado que deve ser escolhido por você.

9. Número máximo de páginas incluindo a capa: 10

A documentação deve conter a descrição do seu trabalho em termos funcionais, dando foco nos algoritmos, estruturas de dados e decisões de implementação importantes durante o desenvolvimento.
Evite a descrição literal do código-fonte na documentação do trabalho.
Dica: Sua documentação deve ser clara o suficiente para que uma pessoa (da área de Computação ou não) consiga ler, entender o problema tratado e como foi feita a solução.
A documentação deverá ser entregue como uma atividade separada designada para tal no minha.ufmg. A entrega deve ser um arquivo `.pdf`, nomeado `nome_sobrenome_matricula.pdf`, onde nome, sobrenome e matrícula devem ser substituídos por suas informações pessoais.

4.3 Código

Você deve utilizar a linguagem C ou C++ para o desenvolvimento do seu sistema. O uso de estruturas pré-implementadas pelas bibliotecas-padrão da linguagem ou terceiros é terminantemente vetado. Você DEVE utilizar a estrutura de projeto abaixo junto ao Makefile:

```
– TP
    |– src
    |– bin
    |– obj
    |– include
    Makefile
```

A pasta **TP** é a raiz do projeto; **src** deve armazenar arquivos de código (*.c, *.cpp, ou *.cc); a pasta **include**, os cabeçalhos (headers) do projeto, com extensão *.h, por fim as pastas **bin** e **obj** devem estar vazias. O Makefile deve estar na raiz do projeto. A execução do Makefile deve gerar os códigos objeto *.o no diretório **obj** e o executável do TP no diretório **bin**. O arquivo executável **DEVE** se chamar **tp1.out** e deve estar localizado na pasta **bin**. O código será compilado com o comando:

```
make all
```

O seu código será avaliado através de uma **VPL** que será disponibilizada no moodle. Você também terá à disposição uma VPL de testes para verificar se a formatação da sua saída está de acordo com a requisitada. A VPL de testes não vale pontos e não conta como trabalho entregue. Um pdf com instruções de como enviar seu trabalho para que ele seja compilado corretamente estará disponível no Moodle.

5 Avaliação

- Corretude na execução dos casos de teste - (15% da nota total)

- Indentação, comentários do código fonte e uso de boas práticas - (10% da nota total)

- Conteúdo segundo modelo proposto na seção **Documentação**, com as seções detalhadas corretamente - (20% da nota total)
- Definição e implementação das estruturas de dados e funções - (10% da nota total)
- Apresentação da análise de complexidade das implementações - (10% da nota total)
- Análise experimental - (30% da nota total)
- Aderência completa às instruções de entrega - (5% da nota total)

Se o programa submetido **não compilar**[2] ou se compilar mas não passar em **pelo menos um caso de teste**, seu trabalho não será avaliado e sua nota será **0**. Trabalhos entregues com atraso sofrerão **penalização de** 2^{d-1} pontos, com d = dias úteis de atraso.

6 Considerações finais

1. Comece a fazer esse trabalho prático o quanto antes, enquanto o prazo de entrega está tão distante quanto jamais estará.
2. Leia atentamente o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.
3. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
4. Plágio é crime. Trabalhos onde o plágio for identificado serão **automaticamente anulados** e as medidas administrativas cabíveis serão tomadas (em relação a todos os envolvidos). Discussões a respeito do trabalho entre colegas são permitidas. É permitido consultar fontes externas, desde que exclusivamente para fins didáticos e devidamente registradas na seção de bibliografia da documentação. **Cópia e compartilhamento de código não são permitidos.**

7 FAQ (*Frequently asked Questions*)

1. Posso utilizar qualquer versão do C++? NÃO, o corretor da VPL utiliza C++11.
2. Posso fazer o trabalho no Windows, Linux, ou MacOS? SIM, porém lembre-se que a correção é feita sob o sistema Linux, então certifique-se que seu trabalho está funcional em Linux.
3. Posso utilizar alguma estrutura de dados do C++ do tipo Queue, Stack, Vector, List, etc? NÃO.
4. Posso utilizar smart pointers? NÃO.
5. Posso utilizar o tipo String? SIM.
6. Posso utilizar o tipo String para simular minhas estruturas de dados? NÃO.
7. Posso utilizar alguma biblioteca para tratar exceções? SIM.

[2]Entende-se por compilar aquele programa que, independente de erros no Makefile ou relacionados a problemas na configuração do ambiente, funcione e atenda aos requisitos especificados neste documento em um ambiente Linux.

8. Posso utilizar alguma biblioteca para gerenciar memória? SIM.
9. As análises e apresentação dos resultados são importantes na documentação? SIM.
10. Os meus princípios de programação ligados a C++ e relacionados a engenharia de software serão avaliados? NÃO.
11. Posso fazer o trabalho em dupla ou em grupo? NÃO.
12. Posso trocar informações com os colegas sobre os fundamentos teóricos do trabalho? SIM.
13. Posso utilizar IDEs, Visual Studio, Code Blocks, Visual Code, Eclipse? SIM.