

---

## Pode isso, Produção?

### 1 Introdução

No cenário industrial contemporâneo, a automação tem se consolidado como um elemento fundamental para a otimização de processos produtivos, garantindo eficiência, precisão e redução de custos. A incorporação de robôs nas linhas de produção exemplifica essa transformação, permitindo a execução de tarefas repetitivas e complexas com alta precisão e adaptabilidade. Para os engenheiros de produção, compreender e controlar essas tecnologias é essencial para desenvolver soluções inovadoras e melhorar a competitividade das empresas no mercado global.

Dado o cenário atual, a empresa PiP GEP percebeu que constantemente funcionários não conseguiam guardar peças não utilizadas a tempo de outro funcionário conseguir usá-la, sendo assim os diretores te contrataram para você desenvolver um sistema capaz de controlar múltiplos robôs pela sua linha de produção coletando objetos que possam vir atrapalhar ou retardar o desenvolvimento da sua cadeia de produção.

### 2 Especificações

A PiP GEP fez o mapeamento prévio das regiões que seus robôs devem explorar. Os dados são expressos numa matriz, conforme a exemplificada na Figura 1. Nela, o caractere ‘P’ indica a presença de uma peça; ‘O’, um espaço inatingível (obstáculo); ‘F’, um funcionário trabalhando; e o caractere ‘.’ (ponto) é uma região vazia. Cada posição é representada por um índice ‘(i,j)’, no qual ‘i’ representa a **linha** e ‘j’, a **coluna**. A origem ‘(0,0)’ é o **canto superior esquerdo**, onde seus robôs partirão e formarão uma base (representada por ‘B’).

```
B . . . . . O O O .  
. . . . . O O O .  
. . . . P O O O P .  
. . . P O O O O P .  
. F . O O O O P . .  
. . . P O O P . . .  
. . . . . . . . .  
. . . F P P . . . .  
. . . F P P . . . .  
. . . . F . . . . .
```

Figura 1: Mapa obtido através dos dados da PiP GEP

Seu sistema também receberá uma lista de comandos a executar, categorizados em dois tipos: **ordens de comando**, e **comandos diretos**. As ordens de comando são destinadas somente aos robôs e NÃO são executadas no momento que são recebidas. Em vez disso, elas são acumuladas no robô de destino. Caso um robô receba uma ordem direta de **EXECUÇÃO**, ele deverá executar **TODAS** as ordens acumuladas. As ordens de comando seguem a seguinte política: *a primeira ordem que chegar será a primeira a ser executada*.

## 2.1 Ordens de comando

Cada ordem de comando é direcionada a um robô específico que é representado por um inteiro positivo 'k' (que pode assumir qualquer valor começando por 0). Todas as **ordens de comando** recebidas por um robô são acumuladas com política já citada e são descritas a seguir. A cada comando executado pelo robô, ele deverá registrar no seu histórico a ação executada, de acordo com o formato específico para cada comando. O histórico deverá ser impresso apenas quando o comando 'RELATORIO' (que será explicado mais adiante) for emitido.

Algumas ordens de comando podem ser enviadas com prioridade, sinalizada por um asterisco '\*' antes do nome do comando. **Uma ordem desse tipo tem prioridade sobre as demais e será, portanto, a próxima a ser executada.**

- 'MOVER k (i,j)': Ao executar esse comando o robô 'k' se moverá até a posição '(i,j)' indicada.

Algumas observações devem ser notadas:

– Vários robôs diferentes podem ocupar uma mesma posição do mapa, mesmo que no local haja uma peça ou um funcionário trabalhando.

– Você deve assumir que o robô se move para a posição designada automaticamente. Dessa forma, não é necessário calcular caminhos, nem planejar trajetórias da posição atual até a indicada.

– Caso exista um obstáculo na posição '(i,j)', o robô deverá registrar em seu histórico:

ROBO k: IMPOSSIVEL MOVER PARA (i,j)

– Caso não exista um obstáculo na posição '(i,j)', o robô deverá registrar em seu histórico:

ROBO k: MOVEU PARA (i,j)

- 'COLETAR k': Ao executar esse comando o robô 'k' tentará coletar a peça na posição em que se encontra se possível. Algumas observações devem ser notadas:

– Caso não existam peças na posição '(i,j)' o robô deverá registrar em seu histórico:

ROBO k: IMPOSSIVEL COLETAR PEÇA EM (i,j)

– Caso existam peças na posição '(i,j)' o robô deverá somar UMA unidade a seu contador de coleta e registrar em seu histórico:

ROBO k: PEÇA COLETADA EM (i,j)

– Recursos coletados devem ser removidos do mapa e substituídos pelo caractere . (ponto).

## 2.2 Ordens diretas

**Todas** as ordens diretas são enviadas diretamente da base para os robôs e têm prioridade sobre ordens de comando, ou seja, mesmo que ainda haja comandos acumulados na fila no robô ‘k’, ele deverá ignorá-los, obedecer imediatamente à ordem direta e imprimir a saída correspondente.

- ‘ATIVAR k’: Envia o robô ‘k’ para explorar o mapa.
  - Caso o robô já tenha saído para explorar, a base deverá imprimir na saída:  
BASE: ROBO k JA ESTA EM MISSAO
  - Caso o robô não esteja em missão, a base deverá imprimir na saída:  
BASE: ROBO k SAIU EM MISSAO – Uma vez que a base será sempre construída na localização identificada pela origem do mapa, todos os robôs ativados iniciam suas atividades de exploração no ponto ‘(0,0)’.
- ‘EXECUTAR k’: Faz com que o robô ‘k’ execute todas as ordens de comando acumuladas.
  - Caso o robô não esteja em uma missão, a base deverá imprimir na saída:  
BASE: ROBO k NAO ESTA EM MISSAO
  - Caso contrário, não é necessário emitir uma saída neste ponto, pois as ordens de comando executadas serão emitidas no relatório.
- ‘RELATORIO k’: Faz com que o robô ‘k’ imprima na saída o seu histórico.
- ‘RETORNAR k’: Faz com que o robô ‘k’ volte para a base.
  - Ao retornar para a base, a base deverá imprimir na saída quantas peças foram coletados:  
BASE: ROBO k RETORNOU PEÇAS y
  - Caso o robô não esteja em uma missão, a base deverá imprimir na saída:  
BASE: ROBO k NAO ESTA EM MISSAO
  - O **histórico** de comandos também deve ser **apagado**.
  - As peças coletadas pelo robô ‘k’, devem ser transferidas para a base. Os contadores do robô devem ser zerados ao retornar para a mesma.

### Atenção:

Ao final da execução de todos os comandos, a base deverá imprimir na saída um relatório geral informando quantas peças foram coletados:

BASE: TOTAL DE PEÇAS y

É importante notar que esses valores representam quantidades totais, a soma de **todos os robôs**.

## 3 Entrada

A entrada é dividida em dois arquivos: ‘**mapa.txt**’, que contém o mapa, e ‘**comandos.txt**’, que contém todos os comandos que devem ser executados pela base e pelos robôs de exploração.

### 3.1 'mapa.txt'

No arquivo de entrada 'mapa.txt', a primeira linha contém a altura e a largura do mapa, isto é, a quantidade de linhas e colunas, separadas por um espaço. A partir da linha seguinte, cada linha do arquivo corresponde a uma linha do mapa, com as colunas separadas por um espaço. Na Figura 2, encontra-se exemplificado um arquivo 'mapa.txt' para um mapa de tamanho  $10 \times 10$ .

```
10 10
B . . . . . 0 0 0 .
. . . . . 0 0 0 .
. . . . P 0 0 0 P .
. . . P 0 0 0 0 P .
. F . 0 0 0 0 P . .
. . . P 0 0 P . . .
. . . . . . . . .
. . . F P P . . . .
. . . F P P . . . .
. . . . F . . . . .
```

Figura 2: Exemplo do arquivo para o mapa de entrada

### 3.2 'comandos.txt'

O arquivo 'comandos.txt' contém todos os comandos diretos e ordens de comando executadas pela base e robôs, distribuídos um por linha.

Como exemplificado na Figura 3, a sequência ativa o robô '0' na posição '(0,0)' com o comando direto 'ATIVAR 0'. Posteriormente, é enviado uma sequência de ordens de comando. A ordem de comando 'MOVER 0 (1,1)' é executada assim que o comando direto 'EXECUTAR 0' é lido. É importante notar que a ordem de execução dos comandos '\*COLETAR 0' e '\*MOVER 0 (2,4)' será alterada pois estão sendo chamados com prioridade, assim, ao entrarem na lista de comandos os mesmos entram na frente dos que já estiverem lá. Desta forma, quando o comando 'MOVER 0 (2,4)' entrar, o mesmo irá para o início da lista. Em seguida, o comando 'COLETAR 0' entrará na frente do 'MOVER 0 (2,4)'. Assim, quando o comando 'EXECUTAR 0' for chamado, o comando 'MOVER 0 (2,4)' será executado, seguido pelo comando 'COLETAR 0' e então os comandos 'MOVER 0 (0,6)' e 'MOVER 0 (4,1)' serão executados.

O comando 'RELATORIO 0' solicita a impressão do histórico do robô 0. Por fim, 'RETORNAR 0' faz com que a base informe quantas peças foram coletadas pelo robô '0'.

```

ATIVAR 0
MOVER 0 (1,1)
EXECUTAR 0
MOVER 0 (0,6)
*COLETAR 0
*MOVER 0 (2,4)
MOVER 0 (4,1)
EXECUTAR 0
RELATORIO 0
RETORNAR 0

```

Figura 3: Arquivo de exemplo com as ordens de comando e os comandos diretos a serem executados

## 4 Saída

Utilize a **saída padrão** ('stdout') para imprimir todas as informações. A saída deve conter a resposta de cada comando, conforme explicado na Seção 2 (Especificações). Por exemplo, para o mapa da Figura 2 e comandos da Figura 3, obtém-se a saída ilustrada na Figura 4.

```

BASE: ROBO 0 SAIU EM MISSAO
ROBO 0: MOVEU PARA (1,1)
ROBO 0: MOVEU PARA (2,4)
ROBO 0: PEÇA COLETADA EM (2,4)
ROBO 0: IMPOSSIVEL MOVER PARA (0,6)
ROBO 0: MOVEU PARA (4,1)
BASE: ROBO 0 RETORNOU PEÇAS 1
BASE: TOTAL DE PEÇAS 1

```

Figura 4: Saída esperada para uma execução cujas entrada são o mapa apresentado na Figura 1 e comandos apresentados na Figura 2

## 5 Observações importantes e dicas

Eis a seguir uma lista de aspectos de implementação relevantes que merecem sua atenção:

- Robôs podem ocupar a mesma posição de outros robôs e peças.
- Robôs não executam ordens de comando imediatamente após recebê-las. Elas são acumuladas conforme a ordem de entrada e são executadas somente ao receber um comando 'EXECUTA k', em que 'k' é o identificador do robô.

- O relatório de ações do robô só é impresso na saída após o um comando direto de ‘RELATORIO k’, no qual ‘k’ é o identificador do robô.
- Ao terminar a execução, deve-se imprimir o relatório geral da base no seguinte formato: ‘BASE: TOTAL DE PEÇAS y’.
- Após retornar à base em decorrência de uma ordem direta de ‘RETORNO’ , o histórico de ações executadas pelo robô, assim como seus contadores de recursos são limpos e os recursos coletados são transferidos para a base.
- Você deve assumir que seu robô é capaz de se deslocar até as posições designadas instantaneamente, dessa forma **NÃO** é necessário fazer buscas ou computar caminhos.
- Note que as saídas **não utilizam** caracteres acentuados e estão escritas em letras **MAIÚSCULAS**. Preste atenção aos espaçamentos e sinais de pontuação, como parênteses e dois-pontos, a saída deve corresponder **exatamente** ao que está especificado neste documento.
- É **obrigatório** liberar eventuais espaços de memória alocada manualmente, caso utilize ‘malloc / new’, ou funções similares.
- Tome cuidado ao manipular ponteiros para evitar erros de acesso.
- Você não sabe quantos robôs serão ativados em cada entrada e os identificadores destes **NÃO SÃO** sequenciais. **Dica:** use uma lista encadeada para armazenar cada robo quando ativado, assim você irá conseguir encontrá-lo para executar as ordens.
- O mapa é **único** para todos os robôs, logo o estado dele deve ser o mesmo para **todos**.

## 6 Entregáveis

Você deve utilizar a linguagem C++ para o desenvolvimento do seu sistema. É **proibido o uso de qualquer container da STL**, isto é, set, map, list, vector e qualquer outro que exista na biblioteca STL. Utilize apenas as variáveis de tipos primitivos e derivados (qualquer um que não esteja contido nos containers da STL) para criar suas próprias implementações para todas as classes, estruturas, e algoritmos.

Você deve organizar seu código-fonte em arquivos separados conforme a responsabilidade de cada objeto e entidade, com nomenclatura condizentes ao que executam. A fim de padronizar a compilação e o projeto, você **DEVE utilizar** a estrutura de projeto abaixo junto ao ‘Makefile’ disponibilizado no *Moodle*:

```
| - src
| - bin
| - include
| - obj
Makefile
```

Diante disso, você **deve implementar** a partir da estrutura passada, **no mínimo duas regras**:

- **all** - gera a compilação padrão usando o comando **make**

- `clean` - remove os arquivos `*.o` e o executável `run.out`.

Você é livre para criar regras auxiliares caso seja necessário

A pasta `'bin'` deve conter o executável (`run.out`) gerado após a compilação de todo o projeto; a pasta `'obj'` deve conter os arquivos intermediários `*.o` gerados; a pasta `'src'` deve armazenar arquivos de código (`*.cpp`); e `'include'`, os cabeçalhos (headers) do projeto, com extensão `*.hpp`.

O executável do seu programa **deverá se chamar** `run.out`. Seu programa deverá receber como parâmetro na linha de comando (`argv`) duas strings, sendo a primeira o nome do arquivo do mapa a ser utilizado e a segunda o nome do arquivo contendo os comandos:

```
run.out mapa.txt comandos.txt
```

Procure seguir boas práticas de programação, se atente ao nome dos métodos, atributos. Evite comentários, lembre-se que seu código deve ser o mais claro possível.

## 6.1 Submissão

Todos os arquivos relacionados ao trabalho devem ser submetidos na atividade designada para tal no Moodle dentro do prazo estipulado. A entrega deve ser feita em um único arquivo, com nomenclatura `nome_sobrenome_cada_membro_equipe.zip`. Este deve conter a pasta com toda a estrutura passada na seção anterior contendo tanto o código-fonte quanto o `'Makefile'` para compilar o mesmo.

```
nome_sobrenome_cada_membro_equipe
|- src
|- bin
|- include
|- obj
Makefile
```

## 6.2 Entrevistas

Todos serão submetidos a uma entrevista presencial no DCC onde será avaliado se você realmente sabe explicar sua implementação:

- As entrevistas terão duração máxima de **5 minutos** onde nós perguntaremos alguma coisa específica sobre sua implementação ou algum aspecto que julgar necessário.
- As entrevistas irão acontecer nas aulas reservadas como **Trabalho Final** no calendário da disciplina. Vocês serão avisados antecipadamente quais equipes serão entrevistadas em quais dias.
- A tolerância para atrasos no dia da entrevista é de **5 minutos**, após isso, será considerado desistente e o trabalho será zerado.

*Atenção:* neste caso **somente** o trabalho final será zerado e **não** todas as atividades.

## 7 Considerações finais

1. Leia **atentamente** o documento de especificação, pois o descumprimento de quaisquer requisitos obrigatórios aqui descritos causará penalizações na nota final.
2. Os principais aspectos avaliados na correção serão:

- Quantidade de acertos para cada teste realizado.

*Atenção:* o programa será avaliado com casos de testes adicionais que **não** serão repassados para vocês. Você é livre para criar novos testes e **pode** até mesmo compartilhar com a turma esses testes para garantir que todos os casos possíveis serão cobertos pelo seu programa.

- Implementação adequada das estruturas especificadas.
- Boas práticas de implementação e organização do código.
- Ausência de vazamentos de memória.

*Atenção:* para cada caso de teste com vazamento de memória, será **descontado** uma certa pontuação.

3. Você terá este trabalho **zerado** se:
  - Utilizar qualquer container da STL para implementar as estruturas, seja de forma principal ou auxiliar.
  - Seu programa não compilar e/ou executar em um ambiente Linux, o mesmo usado para corrigir as Atividades Práticas.
  - O trabalho não estiver **exatamente** com a estrutura de pastas e com o Makefile conforme solicitado.
4. Certifique-se de garantir que seu arquivo foi submetido corretamente no sistema.
5. Em caso de dúvidas sobre o **entendimento do enunciado**, não hesite em perguntar nos fóruns de discussão ou por e-mail.
6. **Plágio é CRIME.** Trabalho onde o plágio for identificado serão automaticamente anulados juntamente com **todas** as atividades da disciplina. **Cópia e compartilhamento de código não são permitidos.**

**Divirtam-se, Produção!**