Universidade Federal de Mato Grosso

Bacharelado em Ciência da Computação

Semestre: 2024/1

Programação 2

Lista 12

Introdução

Você recebeu o código que que apresenta um erro de *segmentation fault* que precisa ser corrigido. Seu objetivo é usar o GDB para identificar o problema, entender o comportamento do programa e propor uma solução. Além disso, você deve responder a perguntas relacionadas aos comandos utilizados durante a depuração.

Instruções

1. Compilação do Programa

Compile o código com a flag de depuração:

```
g++ -g -o debug_example debug_example.cpp
```

2. Execução do GDB

Inicie o GDB com o programa:

```
gdb debug_example
```

3. Execução do Programa

Execute o programa dentro do GDB:

```
run
```

O programa termina com um *segmentation fault*.

Pergunta 1: O que o comando `run` faz e por que ele é importante no início da depuração?

4. Identificação do Ponto de Falha

Quando o erro ocorrer, use o comando `bt` (*backtrace*) para ver a sequência de chamadas de função que levaram ao erro:

```
bt
```

Pergunta 2: Qual é o propósito do comando `bt`? O que a saída indica sobre o ponto onde ocorreu o erro?

5. Análise das Linhas de Código

Liste as linhas de código ao redor do ponto de erro usando o comando `list`:

```
list
```

Pergunta 3: Quais linhas de código são mostradas ao redor do ponto do erro? Qual comportamento suspeito você identifica?

6. Exame das Variáveis

Use o comando `info` e `print` para inspecionar os valores de variáveis presentes no contexto em que se encontra e no contexto da main:

```
info args
up
info locals
down
```

Verifique o conteúdo de `buffer` e das variáveis envolvidas na operação onde o erro aconteceu utilizando o print, por exemplo:

```
print buffer
```

Pergunta 4: Para que serviram os comandos up e down? Qual a diferença dos comandos info args e info locals?

7. Controle do Fluxo do Programa com `next`

Coloque um *breakpoint* na função `isSymbol_Punct` para observar as variáveis locais:

Reinicie o programa e, quando o *breakpoint* for atingido, use o comando `next` para avançar linha a linha:

```
run
next
```

Pergunta 5: O que o comando `next` faz? Como ele ajuda a identificar o ponto exato onde o erro ocorre?

8. Situação para Uso do `step`

Agora, você deve usar o comando `step`, que entra nas funções chamadas, para observar o comportamento da função `isSymbol_Punct`. Coloque um *breakpoint* antes dela ser chamada e use `step` para verificar detalhadamente a execução da função:

1. Execute o programa até atingir o *breakpoint* e, em vez de usar `next`, use o comando `step` para entrar na função:

```
run
step
```

Pergunta 6: O que o comando `step` faz e como ele difere de `next`? O endereço da variável `word` dentro da função é o mesmo do `word` que está fora da função?

9. Situação para Uso do `watch`

Agora, vamos monitorar uma variável em tempo real para observar mudanças. Para isso, você usará o comando `watch`. Monitore o valor de `j`.

```
watch j
```

1. Continue a execução do programa:

```
continue
```

Pergunta 7: Como o comando `watch` ajuda a monitorar mudanças em variáveis? O que você observa no debugger quando o valor de `j` é modificado?

Pergunta 8: Qual o motivo do erro que está acontecendo no código?