Lista de Exercícios 5 (Avaliativa 4)

Linguagens de Programação

Nome: Danilo Pimentel de Carvalho Costa
Matrícula: 2016058077

Questão 1
(c) O programa em si e os estados

Questão 2

A) V
B) V
C) F
D) F
E) F
F) V
G) V
H) V
I) V
J) V

Questão 3

i) A seguir está a relação entre variável e tipo de memória utilizada:

Nome	**Linha**	**Tipo de memória**
valor_inicial	1	memória estática
valor_intermediario	3	memória estática
valores (valor do ponteiro passado como argumento na função calcula)	5	memória dinâmica (stack)
taxa	6	memória dinâmica (stack)
valores (valor do ponteiro para a memória alocada retornado pela função malloc)	12	memória dinâmica (stack)
valores (memória de fato alocada pela função malloc)	12	memória dinâmica (heap)

ii) A seguir se encontram os valores gerados para a lista:

Posição	Valor
0	25
1	75
2	15

Questão 4

Exemplos de coletores de lixo que podem ser utilizados em linguagens de programação:

- Mark and sweep: marca blocos de memória atualmente utilizados analisando as referências no estado atual do programa (mark). Caso não haja marcação, o bloco é removido (sweep).
- Copying collector: divide a memória em duas partes, e armazena blocos em uma de cada vez. Quando a parte atual está cheia, os blocos são analisados para determinar se estão sendo utilizados. Blocos ativos são copiados para a outra parte, que passa a ser utilizada para armazenamento. Blocos inativos não são copiados e são removidos.
- Reference counting: mantém contadores para cada bloco. Para cada criação de referência para o bloco, incrementa-se o contador em 1. Caso uma referência seja removida, decrementa-se o contador em 1. Quando o contador é 0, o bloco é considerado inativo e é removido.

Na minha opinião, o modelo Reference counting é o mais adequado, pois o custo para a gerência de blocos inativos é distribuído nas operações ao longo do programa, e por isso proporciona mais previsibilidade na execução de um programa em termos de performance. Um exemplo de aplicação desse modelo são os Smart pointers em C++, que cria tais contadores para ponteiros quando decorados propriamente pelo programador. Tal abordagem ainda possibilita a flexibilidade para o programador de decidir se deseja gerenciar o ciclo de vida da memória manualmente ou não.

Questão 7

a)

- Polimorfismo paramétrico explícito, ao usar um tipo paramétrico para o ponteiro que a classe envolve.
- Overloading, ao definir um comportamento diferente para os operadores "&" e "*".

b) Não há problema de memória, por que o destrutor do objeto da classe auto_ptr é chamado quando o escopo da função foo0 é destruído. No destructor, a memória alocada para a string também é liberada. Com as duas liberações de memória, o programa não tem vazamentos e o valgrind não acusa nenhum vazamento de memória.

c) Não há problema de memória, por que o destrutor do objeto da classe auto_ptr é chamado quando o escopo do bloco try na função foo1 é destruído. O processo é o mesmo, liberando a memória tanto do objeto da classe auto_ptr quanto da memória alocada para a string, e assim o valgrind não acusa nenhum vazamento de memória.

d) Há problema de memória, por que a execução do programa nunca chega no comando "delete p;" para liberar a memória alocada para a string. Isso se deve ao fato da exceção lançada na linha anterior, que interrompe a execução do bloco try e inicia o tratamento da exceção no bloco catch. Assim, como a memória nunca é liberada, o valgrind acusa o vazamento de memória.