Robson Novato Lobao - 20.1.4018

1. Explique o funcionamento do mecanismo chamado “call-back function”

A call-back function é um código passado por parâmetro para outro código, tornando a comunicação entre camadas mais fácil.

2. Explique o funcionamento do código contido no arquivo “main.cpp”.

O arquivo main.cpp traz a criação de uma aplicação nativa para windows que conta com uma janela, suas características como título, altura e largura e sua instância para o usuário.

3. O que o código contido no arquivo “main1.cpp” faz?

Ele retira 32 posições de ascii da caracter, ou seja enquanto você colocar caracteres minúsculos ele vai devolver ele maiúsculo, é basicamente um toUpper(), até você colocar o caracter ‘s’ que é a saída do programa.

4. Como este código funciona?

Ele funciona calculando na tabela ASCII qual a posição do caracter e retira 32 posições para corresponder ao mesmo char só que maiúsculo.

5. Por que é necessário utilizar as instruções de “type casting”?

Para evitar a entrada de dados que não correspondem ao que o programa espera, como os chars nesse caso. As funções exigem entrada específicas e o type casting garante que elas venham corretamente.

6. O que significa “0xFF” no código “main2.cpp”?

Representação hexadecimal do número 255

7. O que o primeiro loop do código “main2.cpp” faz e como ele funciona?

Pega o conteúdo da string (ou array de chars) e printa até ter o \0 ou a finalização da frase.

8. O que o segundo loop do código “main2.cpp” faz e como ele funciona?

Printa o número de um array e o endereço de memória onde ele está armazenado, pulando de 4 em 4, tendo em vista que é o tamanho de um int em bytes.

9. O que o terceiro loop do código “main2.cpp” faz? Por que ele não funciona?

Era suposto printar o conteúdo e o endereço mas isso não ocorre porque mudamos o tipo da variável do array, alterando o tamanho de bytes e causando uma confusão na aritmética de ponteiros.

10. O que pode ser concluído sobre a aritmética de ponteiros?

Além de ser um tipo de dados muito exato, ele é sensível a mudanças de tipos internos, sempre tendo que ficar atentos ao tamanho em bytes do tipo de dado que estamos apontando, por mexer na memória interna e podendo causar vazamentos, se mal gerenciada.

11. O código possui duas diretivas de pré-compilação que fazem com que o código “main3.cpp” gere erros. Compile e execute o código com a diretiva ERRO1 ativada e desativada e, então, baseado na saída impressa na tela, explique o que o trecho de código de linha 27 a 32 faz

O algoritmo atribui o endereço de memória listado para a variável 'pInt' e imprime o endereço de memória de 'pInt', o endereço de memória para o qual ele aponta e o conteúdo dentro do endereço de memória para o qual ele aponta. Depois disso há uma tentativa de mudar o valor de pInt, porém ‘66’ não é um valor válido para um endereço de memória usando a base hexadecimal.

12. O código possui duas diretivas de pré-compilação que fazem com que o código “main3.cpp” gere erros. Compile e execute o código com a diretiva ERRO2 ativada e desativada e, então, baseado na saída impressa na tela, explique o funcionamento do trecho de código entre as linhas 37 e 57.

Neste trecho temos duas possíveis execuções dependendo da utilização ou não da diretiva ERROR2 primeiramente, a variável 'pInt' é inicializada com NULL, e então, ocorre a impressão de se o ‘ERROR2’ está definido, o endereço da variável 'pInt' e o ponteiro para o qual ela aponta, após isso, 'pInt' começa a apontar para outro endereço de memória e é impresso o endereço de 'pInt' e seu conteúdo, sempre verficando se as variáveis são NULL ou não. Após isso, caso ‘ERROR2’ não esteja definido, tenta-se alterar o conteúdo de 'pInt', que gera um erro, pois o caractere '7' não é um endereço de memória válido. Trazendo como diferença a inicialização ou não do pInt.