1-

São funções passadas como argumento para outras funções, são frequentemente usadas em programação assíncrona para lidar com tarefas como solicitações de rede, eventos de usuário e timers, assim, dando flexibilidade ao código.

2-

O programa se baseie em abrir uma janela com funções da windows.h, usa uma função call-back para processar mensagens recebidas pela janela, de modo que, sempre que ocorrer um evento dentro da janela a função call-back será chamada. O programa usa do recurso de laço “while”, para rodar enquanto não receber a mensagem resposável por “destruir” a janela.

3-

Faz o print de “Hello World”, logo após executa 1 cin e 2 cout contidos em um while. O segundo cout mostra o caractere digitado pelo usuário como letra maiúscula.

4-

Detalhadamente o código funciona da seguinte maneira, é “printada” a mensagem “Hello World”, usando a função padrão“cout”, logo após, são declaradas duas variáveis, a primeira é uma variável do tipo char chamada de “c”, já a segunda é uma variável do tipo inteiro chamada de “incr”. Contudo, é inicializado um while, que será executado enquanto o caractere da variável “c”, seja diferente de “s”. Dentro do while a função cin é chamada, recebendo um caractere para a variável “c”, logo após, é feito um cout mostrando a diferença dentre os caracteres ‘A’ e ‘a’, sendo este valor armazenado na variável incr. Um último cout mostra o caractere digitado em letra maiúscula. Ao sair do ehile temos um system pause, que esperará o usuário digitar uma tecla, logo após termina a execução do programa.

5-

Para printar o valor da diferença de ‘A’ - ‘a’, caso não houvesse o termo (int) à frente da operação, ela não faria sentido, pois são caracteres e não números. O mesmo ocorre ao querer printar o caractere. São feitos castings de tipos.

6-
Indica o caractere 255.

7-

O loop executará enquanto o caractere apontado por pChar seja diferente de ‘\0’, ou melhor, dentro do loop será feito o print de todos os caracteres apontados por pChar, após o print pChar apontará para a próxima casa.

8-

O loop funciona da seguindo a lógica do primeiro, porém agora com inteiros e o cout printará o número apontado por pInt e depois printará o endereço de memória de pInt, dessa forma printando toda sequência de 0 a 9. Além da diferença do while e o do while, que primeiro executará para depois verificar se deve continuar.

9-
Está errado, pois a tentativa de converter um ponteiro de int para um ponteiro de char não faz sentido, já que inteiros e caracteres têm tamanhos diferentes de memória.

10-

Há prós e contras, podendo ser uma “mão na roda”, mas por outro lado pode ser muito perigosa a utilização. Deve ser usada com muita cautela, já que erros podem gerar uma grande dor de cabeça, pois tratar de apontadores é uma tarefa complicada, porém bem útil.

11-

Esta parte do código cria um escopo local usando chaves {} e declara um ponteiro pInt para um inteiro não assinado (unsigned int) e o inicializa com o endereço de memória 0x40300c. Em seguida, imprime o endereço de pInt, o valor apontado por pInt e o valor de variável (que pode ser estática ou local, dependendo da configuração de ERROR1).

Independentemente de ERROR1, o ponteiro pInt é usado para apontar para o endereço de memória 0x40300c, que é tratado como um valor inteiro não assinado. Em seguida, o código imprime o valor do ponteiro pInt e o valor apontado por ele (que pode ser diferente dependendo do que está armazenado em 0x40300c). Em seguida, o valor apontado por pInt é alterado para 66. Por fim, o código imprime o endereço e o valor de variável.

12-

Com ERROR2 ativado, o trecho de código cria um ponteiro não inicializado pInt. Ele verifica se pInt é válido usando a expressão (pInt), que verifica se pInt não é nulo (ou seja, se aponta para algum endereço de memória).

No entanto, como pInt não foi inicializado, a primeira verificação (pInt) resultará em "FALSE", indicando que o ponteiro não é válido. A segunda verificação (pInt != NULL) também resultará em "FALSE" pelas mesmas razões.

Em seguida, o código atribui explicitamente um endereço de memória (0x40300c) a pInt. Após essa atribuição, pInt agora aponta para um endereço de memória válido, e a terceira verificação (pInt != NULL) resultará em "TRUE".

om ERROR2 desativado, o trecho de código inicia pInt com NULL (um ponteiro nulo). As verificações (pInt) e (pInt != NULL) resultarão em "FALSE" inicialmente.

Depois, pInt é explicitamente configurado para apontar para o endereço de memória 0x40300c. Agora, pInt é válido e as verificações subsequentes (pInt != NULL) resultarão em "TRUE"