Контрольні запитання програмування

Ветріченко Дарина

1. Модульний тест — це автоматизований тест, що перевіряє окремий елемент програми (функцію, клас) ізольовано від інших.
2. Щоб протестувати функцію конкатенації двох рядків, треба викликати її з відомими рядками і перевірити, чи результат дорівнює очікуваному.
3. Для тесту функції об’єднання двох масивів треба передати два масиви, перевірити, чи результат містить всі елементи обох масивів у правильному порядку і має правильний розмір.
4. Модульний тест запускається через тестову програму або відповідну команду (наприклад, make test).
5. Модульні тести виконуються після зміни коду, додавання функцій або при автоматичній перевірці якості (CI/CD).
6. Принципи модульного тестування: ізольованість, незалежність, простота, автоматизація, повторюваність, чіткість результату.
7. Модульні тести потрібні для раннього виявлення помилок, контролю якості, спрощення змін у коді. Ще бувають інтеграційні, системні, регресійні, приймальні та навантажувальні тести.
8. Покажчик на масив створюється як int\* p = arr або int\* p = &arr[0].
9. До покажчиків можна застосовувати арифметику, розіменування, порівняння, присвоєння.
10. Пам’ять звільняється через delete, delete[], або free у C.
11. Виділення пам’яті здійснюється через new, new[], або malloc у C.
12. Контроль за витоком пам’яті досягається правильним звільненням пам’яті, використанням інструментів типу valgrind, смарт-покажчиків.
13. Статичний масив має фіксований розмір і виділяється на стеку, динамічний — змінний розмір і виділяється у купі.
14. Обсяг для динамічного масиву треба зберігати окремо, бо sizeof(p) поверне розмір покажчика.
15. У фрагменті pointMas = &mas[0]; for(i = 0; i < arraySize; \*pointMas++ = i++); — у масив mas записуються числа від 0 до arraySize-1.
16. Динамічна пам’ять підлягає звільненню, інакше виникне витік пам’яті та перевитрата ресурсів.
17. Текст на консоль виводиться через std::cout << "текст".
18. Дані з клавіатури зчитуються через std::cin >> змінна або getline(std::cin, рядок).
19. Форматований вивід дозволяє задавати вигляд даних, неформатований — виводить як є.
20. Прототипи функцій введення/виведення у файлах iostream, fstream, stdio.h.
21. Для адреси змінної використовується символ &.
22. Форматування даних при виведенні робиться через маніпулятори setw, setprecision або printf у C.
23. Функція читання з клавіатури зчитує дані у змінну і очищає буфер.
24. Функція виведення передає дані у потік виведення (на екран, у файл).
25. Текст з декількох слів вводиться через getline(std::cin, рядок).
26. Текст з декількох слів виводиться через std::cout << рядок.
27. Два рядки склеюються через оператор + або функцію strcat у C, у C++ — s3 = s1 + s2.
28. Підрядок у рядку шукається через strstr у C або find у C++.
29. Рядки у C називають “Null-terminated”, бо завершуються нуль-символом \0.
30. Два рядки порівнюються через strcmp у C або == у C++.
31. Щоб видалити підрядок, треба знайти його позицію (find) та видалити (erase) у C++.
32. Файл — іменована область пам’яті для зберігання даних на носії.
33. Для неформатованого введення є fread, fgetc, fgets, read.
34. Для неформатованого виведення — fwrite, fputc, fputs, write.
35. Розмір файлу визначають через fseek/ftell або tellg у C++.
36. Коли кількість даних невідома, читати до кінця файлу у циклі (while(!eof())).
37. Дані при записі у файл можна форматувати через fprintf або << з маніпуляторами.
38. Рядки з файлу зчитують через getline, записують через <<.
39. Символи — через fgetc, fputc у C, get/put у C++.
40. Текстові файли містять символи, бінарні — “сирі” байти; у текстових може бути перетворення символів.
41. Доступ до елементів структури — через крапку (obj.field) або стрілку для покажчика (ptr->field).
42. Можна присвоїти одну структуру іншій того ж типу оператором присвоєння.
43. Покажчики на структури потрібні для передачі у функції, роботи з великими об'єктами.
44. Членом структури може бути інша структура.
45. В об’єднанні всі поля займають одну область пам’яті, у структурі — окрему.
46. Бітові поля дозволяють задавати розмір у бітах, економлять пам’ять.
47. Перерахування (enum) — тип, що приймає одне з фіксованих значень.
48. Суміш (union) — тип, що дозволяє зберігати в одному полі значення різних типів, але тільки одне одночасно.
49. Додавати елементи до динамічного масиву можна через перевиділення пам’яті, копіювання у більший масив.
50. Структура для динамічного масиву: масив-покажчик, змінна розміру, змінна місткості.
51. Динамічний масив може змінювати розмір, звичайний — ні.
52. Додавання елемента — розширити масив при потребі, додати у arr[size++].
53. Видалення — зсунути елементи, зменшити size.
54. Доступ до елемента списку за номером — пройти список у циклі, рахуючи індекс.
55. Нові елементи додають створенням нового вузла і зміною покажчиків.
56. Головний елемент зберігає адресу початку списку.
57. Кінець списку — покажчик next дорівнює nullptr.
58. У односпрямованому списку один покажчик, у двосписку — два.
59. Для односпрямованого: struct Node { int data; Node\* next; }
60. Список — динамічна структура з вузлів, зв'язаних покажчиками.
61. Головний елемент потрібен для швидкого доступу до початку списку.
62. Поміняти місцями два елементи — змінити їхні покажчики next/prev.
63. Переваги списку: динамічний розмір, швидке додавання/видалення; недоліки — повільний доступ за індексом, більша витрата пам’яті.
64. Для двосписку: struct DNode { int data; DNode\* next; DNode\* prev; }
65. Двосписок дозволяє переміщення вперед і назад.
66. Головний і хвостовий елементи — для швидкого доступу на початок і кінець.
67. Перехід від голови до хвоста: for(Node\* p = head; p; p = p->next) {...}
68. Від хвоста до голови: for(DNode\* p = tail; p; p = p->prev) {...}
69. У односписку 1 покажчик, у двосписку — 2.
70. Лінійний список — послідовний, нелінійний — дерево, граф.
71. Кільцевий двосписок — останній вузол вказує на перший, і перший на останній.
72. При сортуванні списків змінюють порядок покажчиків, а не копіюють дані — це ефективніше.
73. У списку може бути кілька покажчиків (мультисписок) — для різних зв’язків.
74. Клас — тип даних з атрибутами і методами; від структури відрізняється інкапсуляцією (за замовчуванням доступ private).
75. Метод — функція класу; від функції відрізняється тим, що працює з даними об'єкта.
76. Інкапсуляція — приховування реалізації і даних від зовнішнього світу.
77. Константні методи не змінюють об’єкт, пишуться з const у заголовку.
78. Розмір об’єкта визначається через sizeof(obj).
79. Права доступу обмежують доступ до атрибутів і методів.
80. Для доступу до приватних атрибутів потрібні геттери і сеттери.
81. Const-методи потрібні для гарантії незмінності об’єкта.
82. Константні методи потрібні для роботи з const-об’єктами.
83. Атрибут може бути константним, якщо оголошено з const і ініціалізовано.
84. Конструктори потрібні для ініціалізації об’єктів.
85. Конструктори відрізняються параметрами і поведінкою.
86. Конструктори можна перевантажувати.
87. Деструктори потрібні для звільнення ресурсів.
88. Деструктор не має параметрів, не повертає значення і викликається автоматично.
89. Конструктор копіювання створює копію об’єкта, потрібен для коректного копіювання складних об’єктів.
90. Конструктори викликаються при створенні, деструктор — при знищенні об’єкта.
91. Списки ініціалізації — це спеціальний синтаксис для ініціалізації полів до виконання тіла конструктора.
92. Перевантаження методів — це оголошення методів з однаковим ім’ям, але різними параметрами.
93. Виклик перевантажених методів визначається кількістю і типами параметрів.
94. Виведення даних через потоки — std::cout << ...
95. Читання даних — std::cin >> ... або getline(std::cin, ...)
96. Stringstream потрібен для парсингу і форматування рядків як потоків.
97. String у C++ — клас для безпечної роботи з рядками, на відміну від char\* у C.
98. Простір імен — це механізм групування імен для уникнення конфліктів.
99. Вивід у текстовий файл — через std::ofstream fout("file.txt"); fout << ...
100. Читання з файлу — std::ifstream fin("file.txt"); fin >> ...
101. Вивід у бінарний файл — через ofstream з ios::binary та функцію write.
102. Маніпулятори setw(w) і setprecision(d) задають ширину поля і точність; треба підключити <iomanip>.
103. Текстові файли зберігають дані як символи, бінарні — як байти; у текстових можливе перетворення символів.
104. Оператори потрібні для виконання операцій над даними.
105. Перевантажувати можна майже всі оператори, крім деяких службових.
106. Не можна перевантажити ::, ., .\*, ?:, sizeof, typeid, alignof, static\_cast.
107. Серіалізація — це перетворення об’єкта у потік байтів, десеріалізація — зворотній процес.
108. Оператори перевантажують через функції-оператори у класі.
109. Спадкування потрібне для повторного використання коду і створення ієрархій.
110. Права доступу визначають, які атрибути та методи будуть доступні у нащадках.
111. Атрибути при спадкуванні — public, protected, private — впливають на доступність у наслідниках.
112. Спадкування працює для об’єктів-нащадків, вони мають усі поля і методи предка (крім приватних).
113. Ієрархія класів — це структура, де класи пов’язані відношенням “є-нащадком”.
114. Ієрархії бувають деревоподібні, ланцюгові, “ромбоподібні” (множинне спадкування).
115. Спадкування — це “є типом”, агрегація — “містить”.
116. Поліморфізм — здатність викликати різні методи через один інтерфейс.
117. Віртуальні методи потрібні для поліморфізму.
118. Чисто віртуальні методи — це методи з =0 у оголошенні, без реалізації у базі.
119. Таблиця віртуальних методів — структура для швидкого виклику віртуальних функцій.
120. Абстрактний клас — клас з хоча б одним чисто віртуальним методом.
121. Об’єкт абстрактного класу створити не можна.
122. Інтерфейс — клас, у якого всі методи чисто віртуальні.
123. Для поліморфізму потрібні віртуальні методи і покажчик/посилання на базовий клас.
124. Абстрактний клас може містити реалізацію і дані, інтерфейс — лише чисто віртуальні методи.
125. Абстрактний клас не можна створити напряму, звичайний — можна.
126. При перевизначенні віртуальних методів треба співпадіння сигнатури, використовувати override.
127. Final — забороняє подальше перевизначення, override — вказує, що метод перевизначає базовий.