**II. Керування пам’яттю**

1. Купа та стек.
2. Поняття купи. Призначення. Принципи роботи.
3. Стек. Принцип роботи. Реалізація апаратного стеку в оперативній памяті.
4. Стек викликів функцій. Стековий кадр.
5. Розміщення в стеку параметрів функцій, автоматичних змінних, адреси повернення.
6. Вказівники та посилання.
7. Операції над вказівниками. Адресна арифметика. Вказівник типу void\*.
8. Звязок між вказівниками та масивами. Символьні масиви. Масиви вказівників. Аргументи командного рядка.
9. Вказівники на функції.
10. Посилання. Опеарції над посиланнями.
11. Різниця між вказівниками та посиланнями.
12. Робота з памяттю.
13. Функції malloc/realloc/calloc/free.
14. Memory Leaks. ( Інстументи виявлення, Visual Leak Detector ).
15. Параметри та результат функції.
16. Передача параметрів в функції за значенням, через вказівник, через посилання.
17. Передача масиву у функцію.
18. Повернення результатів із функції. Оператор return, поверення результатів через параметри, void-функції.

**III. Основи ООП**

Вступ: Що таке обєкти ? Їх призначення. Порівнння ООП та функціонального підходів до програмування. З чого складається обєкт ( поля, різні для кожного обєкту, та методи – спільні для всіх обєктів ).

1. Області видимості елементів класу. Принцип інкапсуляції. Сеттери і гетери.
2. Inline методи. Розказати, що методи, оголошені inline повинні мати реалізацію в юніті компіляції, з якої використовуються і в результаті inline можна оголошувати або лиш публічні методи з реалізацією прямо в оголошенні класу, або приватні для внутрішнього використання.
3. Конструктори та деструктори.
4. Конструктори, конструктор за замовчуванням, конструктор копіювання. Порядок ініціалізації класу.
5. Список ініціалізації. Порядок виклику. Порівняння списку ініціалізації із присвоєнням в конструкторі.
6. Деструктор. Порядок деініціалізації класу.
7. Виклик конструкторів та деструктора при динамічному виділені памяті. Оператори new, new[], delete, delete[]. Порівняння із функціями стандартної бібліотеки С. Альтернатива nothrow, буферизований оператор new.
8. Переваги використання автоматичних змінних над динамічними ( технологія парних операцій – конструктор/деструктор ).
9. Використання операторів new та delete для полів-вказівників.
10. Виклик деструкторів та конструктора для глобальних обєктів.
11. Неявний виклик конструкторів. Зарезервоване слово explicit.
12. Перевизначення функцій та параметри по замовчуванню.
13. Перевизначення функцій та методів.
14. Параметри по замовчуванню.
15. Перевантаження операцій.
16. Оголошення операторної функції. Бінарні та унарні оператори. Перфіксна та постфіксна форма перевантаження. Стандартні прототипи стандартних операторів ( порівняння, присвоєння, інкременту/декркменту, арифметичних, квадратних дужок, круглих дужок ).
17. Перевантаження в зовнішній функції та дружні функції.
18. Перевантаження оператора присвоєння.
19. Перевантаження operator new та operator delete в класі та глобално.
20. Обмеження на перевантаження операцій.
21. Статичні поля і методи.
22. Поняття статичного поля. Оголошення та інстанціювання як глобальний обєкт.
23. Способи звернення до статичних полів.
24. Обмеження статичних функцій.
25. Застосування статичних членів класу
26. Оголошення класів, структур, енамів в середині класу ( публічне та приватне ).
27. Оголошення класів форвардами :
28. Уникнення зайвого включення h-файлів.
29. Уникнення кругових залежностей при використанні вказівників.
30. Константні поля та методи.
31. Констанні поля, статичні константи, enums. Ініціалізація.
32. Константні методи. Виклик методів константного обєкту.
33. Mutable, const\_cast<> та константні поля.
34. Шаблони. ( Пояснити як вони реалізуються. )
35. Шаблонні функції.
36. Перевантаження та спеціалізація шаблонних функцій.
37. Стандартні параметри в шаблонних функціях.
38. Шаблонні класи. Спеціалізація шаблонних класів. Методи шаблонного классу потрбіно включати в кожен юніт компіляції, що використовує шаблон.
39. Статичні члени в шаблонних класах. Статичні поля шаблону потрібно інстанціювати окремо для кожного набору параметрів, що зустрічається у програмі.
40. Обмеження на шаблони.
41. Розумні вказівники.
42. Поняття та види розумних вказівників.
43. Створення, присвоєння, знищення.
44. Перевантаження операторів розумного вказівника.
45. розумні вказівники, вимагаючи видимості конструктора і деструктора класу, збільшують залежності між заголовочними файлами і не дають використовувати в них оголошення класів форвардами. В результаті вони призводять до більшої зв’язаності модулів і збільшення часу компіляції програми при змінах

**Для чого потрібна купа ?**

Порівняти динамічні обєкти, із створенням та видаленням статичних. Автоматичних обєктів.

Немає зборщика мусору – треба звільняти. (приклад веб сервера)

**Принцип роботи**

Память ділиться на заняту і вільну.

При звільнені функція звільнення повинна знати : розмір виділеної памяті, чи це зайнята чи вільна память. При виділені – які участки вже зайняті.

Реалізується списками. (Попередній, наступний, можна розмір). Ефективність. Фрагментація та злиття.

**Стек** – область призначена для тимчасового зберігання даних

**Принцип роботи стеку**

Вершина стеку

Апаратний стек росте в сторону зменшення адрес.

PUSH / POP і зміна вершини стеку.

**Передача параметрів та повернення значення**

Домовленість про виклики - fastcall (через регістри) – stdcall через стек. Зміна параметру не впливає на оригінал.

**Стек викликів**

Основне призначення – відслідковувати адреси повернення

**Стековий кадр**

При помощи EBP мы можем адресовать + параметри, - локальні змінні.

Змінні завжди не можуть бути статичними ( з прямою адресацією ) – рекурсивні(кожен виклик повинен мати свою унікальну память, при кожному виклику треба виділяти память а потім звільняти) а бо в 2х потоках. Кожного разу змінна має інший адрес, і для доступу є EBP.

Поняття Адресація відносно стекового кадру.

**Розкрутка стеку**

Балансування стеку

**Вказівник**

Оголошення. Скліьки займає памяті. Операції \*, &

& - тільки до змінних ( неможна, вирази, косатанти і регістр.змінні).

\* - операція посилання через вказівник – розіменування.

\* & більший приоритет чим арифметичні – y = \*ip + 1;

Унарні операції \*, ++ -зправа на ліво – (\*ip)++; ++\*ip;

Можна присвоювати одинакові вказівники.

константний вказівник на константні дані. Const char\*

void\* = int\*, int\* != void\*; void\*- немає типу 'void \*' : unknown size

**Вказівники та аргменти функцій**

Swap( int\*, int\* ); void swap( &a, &b );

Заборона змінненя – const int\*

Вигода – важкі обєкти не потрібно копіювати в стек

Можна зберегти результат функції. (квадратне рівняння).

**Вказівники та масиви**

Операцію індексації можна зробити за доп.вказівників, причому так працює швидше.

Операції адресної арифметики працюють не з байтами а з блоками розміру типу. ++, -= .

Порівняння, додавання цілих, різниця вказівників. Присвоєння однотипних, 0. Void\* = new int;

Вказівник змінна, масив – константа!

Символьні масиви закінчуються 0. Перехід по стрічці на один символ while( \*p != 0 ) cout << \*p; strcpy();

Використання вказівників масивів - сортування. char\* aszData[ 10 ];

Аргументи командного рядка – масив вказівників.

**Const та вказівники**

Константний вказівник повинен бути ініціалізований.

**Помилки з вказівиками**

Неініціалізований вказівник.

**Посилання**

Використовується для передачі параметрів у функцію, для повернення результатів, як змінна.

Повинно бути ініціалізоване. Посилання на константу повинне бути константним.

Обмеження: ініціалізуються завжди при створені. Силки не мають адреса, неможна стоврити вказівник на силку.

Не існує нульової силки.

**Робота з динамічною памяттю**

В С++ дві системи – одна з С, інша С++ (краща). Память виділяється з кучі.

Слід включити stdlib.h або cstdlib. Слід приводити тип int\* p = (int\*)(10\*sizeof(int));

Для nothrow заголовок <new>

Якщо два рази видалити той же вказівник – менеджер памяті може зациклитися.

**Відмінність malloc/free та new/delete**

new кидає екзепшин ; operator new (виділяє память) конструктор, деструктор -operator delete .

якщо new[], delete[] – то конструктор/деструктор викликається декілька раз.

Щоб не викликати констр/дестр можна працювати з operator new, operator delete.

**set\_new\_handler**

можна звільнити память і продовжити виконання.

**Перевантаження operator new, operator delete**

Перший параметр має бути size\_t, можуть бути додаткові параметри.

Стандартні зазвичай реалізовані крще ніж кастомні. (Дьюхерст). Краще перевантажити з дод.параметром

**Memory Leak**

Visual Leak Detector

**Placement new**

Щоб жорстко закріпити адресу обєкта. Слід самому викликати деструктор. Слід звільняти память буферу.

Неможна викликати звичайни delete (помилка для автоматичних або статичних даних).