

- Обединение - способность от промежука, която съоделат един и същия начин;
→ union A { int a; char b[3]; }
→ A obj; obj::a;
→ sizeof = размера на най-голямия тип данни
- Endianess - наим на подредбата на байтовете на дадена дума;
→ Little endian - най-старият байт е най-отляво в работния с него
→ Big endian - обратното

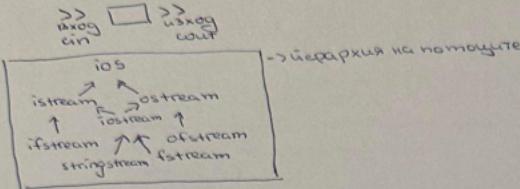
```
bool isLittleEndian() {  
    [44]  
    union Test { char arr[20];  
    int a; } test;
```

```
test.a = 1;  
return test.arr[0] == 1, 4
```

- Задача за № приетственка

Тема №2:

- **намак** - последовательност от байтове, наследие от std::vector.



- потоци за изход
 - ostream (напр. cout), ofstream - за файлове; имат рут-указател
 - форматиран изход:
 - os << <обект>; → връща os
 - синхронизация:
 - flush();
 - позиционирани:
 - tellp() - връща тисло (позиция на рут-указ. в потока);
 - seekp(index) - слага рут-указ. на дадения индекс
 - seekp(offset, std::ios::beg | end);

• потоци за вход

- ifstream (cin), ifstream
- форматиран вход:
 - is >> <обект>;
- позиционирани:
 - tellg(); - връща теге и get-указ.
 - seekg(index) - слага get-указ. на инд.
 - seekg(offset, direction)
- подформатиран вход:
 - get() - връща тисло!
 - read() - за четене на двоични файлове
 - getline(<буквер> дразнер, <разделител>)
 - peek() - връща тисло, но не чете чрез.
 - ignore() - чисти чрез, не връща нищо
 - ignore(брой позиции/разделител)

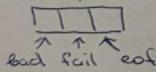
⚠ >>, .getline(), .ignore прескокат \n.

- файл-последователност / масив от байтове (максимум от 0до 255).

• Режими на работа:

- 1 -> ios::in -> отворен поток за четене
- 2 -> ios::out -> за писане
- 4 -> ios::ate -> рут-указ в края, не позволява възпроизв. изч.
- 8 -> ios::append -> рут-указ в края, не е позволено възпроизв. на
- 16 -> ios::trunc -> ако файлът є, се изтрива съдържанието му
- 32 -> ios::binary -> отваря файлът в двоичен формат, третира \n по раз. начин
- 64 -> ios::nocreate -> ако є файлът, то отваря
- 128 -> ios::noreplace -> ако є файлът, то отваря

• Годжини:



- bad() -> показва дали има запушка на иndo.
- fail() -> дали определения е невалиден
- eof() -> дали е достигнат края на файла
- good() -> е успешна
- clear() -> изчиства грешките
- op. eof ≡ !(fail || bad)

• отваряне на поток

```

std::ifstream ifs("именеФайл", <режим>);
if (!ifs.is_open())
    throw"
    
```

```
fs.flush();
fs.clear();
3
fs.seekg(curr);
fs.close();
```

4

Тема №4:

- член-функции - функции в тялото на клас / ~~най-често~~ структурата; работят директно с член-даниите на обекта/структурата, използват се от обект;
→ преобразува се от компилатора до нормална форма, като подава и допълнителен параметър (указват как обекта, от който се използва):
$$\text{struct A} \{ \dots \}$$
$$\text{void g();} \rightarrow \text{void g(A* } \underline{\text{this}} \text{);}$$
$$\text{const void f();} \rightarrow \text{void f(const A* } \underline{\text{const this}} \text{);}$$

△ const. функции могат да се опират без $\#$, независимо във неконст.

- конструктор - член-функция, която се извиква при създаването на обект; използва се за инициализация на данини, има общото име като класа/структурата, не се оказва тип на временните експресии.

- к-р без параметри се нарича default-ен (ако не се различава, компилаторът генерира автоматично)
- к-р само с 1 параметър се нарича конвертиращ (explicit изразяване)
- при $A arr[10]$ се вика 10 пъти default-ен к-р
- при влагане:

$$\text{struct A} \{ \dots \}$$
$$B obj1; \rightarrow A() : B(), C()$$
$$C obj2; \rightarrow B(), C(), A()$$

Ред на викане на к-ри

- деструктор - член-фия, $\#$ само $!$; извиква се при унищожаване/изтриване на обекта; има общото име като клас/структурата, но с \sim от пред, назава рода такъв; използва се за унищожаване на външни ресурси
- при влагане:

$\sim A, \sim C, \sim B$

△ К-ри и дестр. се извикват последователно, защото е възможно дадената висяща единица от друга.

- Абстракция - използване наедре за итересуване как работи
- Капсуляция - ограничаване на достъпа:
 - private → иначе достъп до тези данни само в члените на класа
 - protected → в клас и наследниците му
 - public → $\#$ иначе достъп до тези данни.

• модификатори за состояни

- get-функции - временен копие / конст. реф / указ. като член-даниите
- set-функции - позволяват модификация, но под контрол.

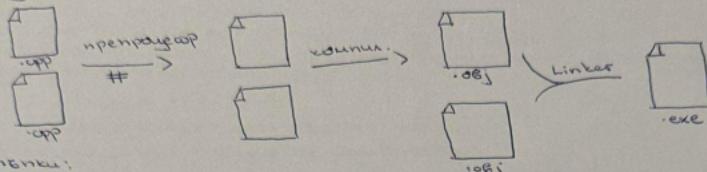
- mutable - член-дани, които могат да се модифицират дори и пред const. функции; не влияят на видимото състояние на обекта

- класове - като структури, но тук no default $\#$ e private.

1. ПОНЯТИЯ

Тема №5:

- разделена компиляция

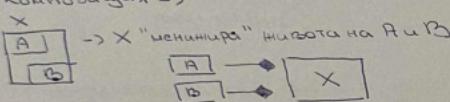


- * Стадии:

- > препроцессор - избавляет предпроцессорные директивы (текстобработка)
- > компиляция - извлекает фрагменты, новые строки, табуляции, которые не являются необходимыми
- > символьический анализ - проверяет строкового формата в окне программы
- > синтаксический анализ - проверяет формат кода и на ошибки
- > оптимизация
- > assembly code
- > linking

- * #include - заменяет #define

- . Компиляция ->



- * Агрегация



но не!!!

- . Копиранет к-р к-р, като приема обект от базовия клас и текущия обект става негово коние; ако него различен експлицитно, компилаторът генерира автоматично; издават нов обект

A (const A& other){
 { методът (this, &other, sizeof(A)), }

-> при влагане к-к на обекта вика к-к на тлен-датните си.

- . Оператор = -> има базовата семантика като к-к., т.е. приема обект от базовия клас и текущия става негово коние, но така се поддържат все съществуващи обекти; ако не се различат, компилаторът генерира

A obj;

A obj2(obj)//к-к.

obj2=obj; //он=

obj3=obj; // к-к.

- . RVO - return value optimisation, спомага за правилото на коний

A createA(){
 A obj;
 //
 return obj;
}

y

Тема №6:

- динамична памет в класовете → при default-но генерираните от компилятора к.к. и он = копиратща shallow copy, т.е. инициализацията на 2 различни обекта е 2 различни указателя, които също са идентични във времето резултат от динамичната памет. Т.е. при присвояване трябва 2 пъти да се прави copy и move. => или изброяване копираниято.

A (const A&) = delete;

A& operator=(const A& other) = delete;

или еквивалентно разписване копираниято:

```
class A {
    int x;
    B* ptr;
public:
    A (const A& other) {
        x=other.x;
        ptr = new B(other.ptr);
    }
    A& operator=(const A& other) {
        if(this!=&other){
            x=other.x;
            ptr=new B(other.ptr); } // логиката може да не е такава
        return *this;
    }
};
```

- Голяма тема въпроска - к.к., def. к-р, gefip, on = → разписване само когато има дин. памет в класа

Tema № 4:

- Видове оператори:
 - > бинарни -> на 1 аргумент (`!, ++, --`)
 - > бинарни -> на 2 аргумент (`+, -, *, /`)
 - ! та е унариен, че бинарен, като та не прави ищущ
 - > тернарен -> `<bool-УСЛ> ? а : б;`
 - Характеристики на операторите
 - 1) ассоциативност
 - > ляво
 - $+,-,*,/$
 - > дясно
 - =
 - 2) приоритет
 - 3) позиция на операндите
 - > префиксни
 - $++$
 - > суфиксни
 - $++$
 - > индексни
 - Предредфиниране на оператори:
 - > `+=, -=, /=, *=` предредфиниране като член-ф-ции, той като времеят пред. ѝ си левият обект
 - > `+, -, *, /` -> всички ф-ции
 - > предредфиниране на оператор `++`:
 - A~~в~~ оператор `++()`; // prefix
 - A~~в~~ оператор `++(int n)`; // suffix с думата параметър
 - приятелски класове и ф-ции:
 - > ф-ции - всички функции, които имат достъп до private и protected член-данините.
 - > класове - всички класове, които имат достъп до private и protected член-данините на текущия клас.
 - предредфиниране на операторите за лявото - в `<< и >>` помощят е левият аргумент, т.е. задължително ги правят всички (по неление - приятелски)
 - оператори, които трябва да са вътрешни $\rightarrow \text{on} =, \text{on}()$
 - оператори, които не можем да предредфинираме \rightarrow тернарен, `::, .`
 - ⚠ Не се предобр. акоу, позиция и приоритет

Тема №8:

- static local variables - съзнат при глобалните променливи, инициализират се веднага при първото влизане в scope-а, уничтожават се в край на изпълнение на програмата.
- static функции - обявяват се в класът, единица (1. срп файл), еквивалент на ф-я в анонимен namespace.
- static член-данини на клас - не се свързват с конкретен обект, класът представява "пакет", в който са "обявени"; в статичните функции няма оператор this, има достъп само до други статични член-данини на класа.
- Исклучения - сигнализират проблем във вид на исключение
- throw (обект) // move к-р | copy к-р, ако има move
 - throw служа инициално д-ръм, започва stack unwinding, т.е. в д-ръм предходна отговорността за грешката на тази надред
 - обработка на грешките - чрез try-catch блок
 - * try { // код }
 catch (std::exception& exc) { // обработка }
 finally { // Винаги се изпълнява }
 - ! Също при try-catch блок се викат дестр на все построени обекти.
- Видове грешки:
 - std::bad_alloc - при неуспешно зарезане на дин. памет
 - std::bad_cast - при касиване
 - std::logic_error - при нарушаване на инвариантите на класа, когато се предотвратят
 - std::runtime_error
 - std::range_error $\xrightarrow{\text{касл}}$
- Извл. в к-р:

```
class A {
public:
    A() { char* arr;
        arr = new char[n]; } // Badalloc
```
- Извл. в дестр: не го правим:

```
class A {
public:
    ~A() { throw b; } //
```
- int g() {
 A obj;
 throw b;
 }
 - throw-ва 2 exception-a, което автоматично вика std::terminate, което също протича
- Ниво на сигурност:
 - no throw guarantee - нами случаи, в които да хъбвам изкл. (делението на 0)
 - strong exception safety - възможност е хъб на изкл, но обекта не изчезва
 - basic exception safety - възможност е хъб на изкл, но обекта остава usable
 - no exception safety - кашо не е гарантирено

Тема № 9:

- Насиви от указатели към обекти :

- > no-бройн свар-объект (нестандарт) (+)
- > инициализация с конст (+)
- > resize не правилно конст (+)
- > инициализация от def-к-р (+)
- > no-инициализация (+)

-> при насиви от обекти инициализация locality, т.е. при четене и изтегляне чрез данните етапи по-бързо, тъй като са подгответи междувременно в паметта.

- Move-семантика

- > типове данни :

-> prvalue (литерали - 4, false, nullptr; инициализации на др-а, които временно конст) + xvalue (обекти, за които инициализацията е която временно) = rvalue

-> lvalue - към имена на променливи, ф-ции, лок-данини, ф-а, които временно рефериранция

f(int a) // lvalue и rvalue

f(int&a) // lvalue

f(const int&a) // lvalue и rvalue

f(int&&a) // rvalue ref, не прави конст, директно крае
данните с move

-> no-lv - не се правят инициализации, а се "извадят" данните на
друг обект, нести време и памет

-> std::move - преобразува lvalue в rvalue, индицира края на
инициализацията на обект

-> move x = p // no- : с тях не се освобождават временни обекти

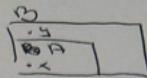
-> A&& се третира като lvalue, докато не се приложи std::move

Tema № 10:

• Наследяване - is-a-relationship

→ дефиниране клас, който има имената на базовото наследение и член-датни като базови член-датни

class A {
 int x
}
class B : A {
 int y;
}



→ Наследяват се методи и член-датни, когато се ползват със `super` go protected и public член-датните

→ Видове наследяване:

- > private: protected и public член-датните са private, ако private давате имена със `super`
- > protected: protected и public член-датните са protected, ако protected давате имена със `super`
- > public: & protected и public са остават, ако public давате имена със `super`

→ в class наследен по default е private, в struct - public

→ параметри на дъщери - Base; Der: Base

f1 (Base)	g1 (Der1)
f2 (const Base)	g2 (Der2)
f3 (const B*)	g3 (Der3)
<u> </u>	<u> </u>
" der, и Base, запомняте имената на Der и от Base f (Base* arr) -> не негативне наименв от Der !	како der

→ k-p -> в k-p яка Der получава извиквана k-p на Base, ако не е западено тои, сеника default-ен; никога извиква нещо Base

→ copy -> извиквана функция копир. на базова база

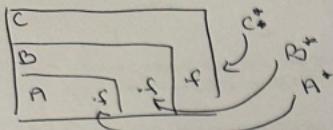
→ конструиране -> copyFrom се грижи само за датните на клас-наследник. Трябва да вникнеш експлицитно k-k; B on = обикновено on = на базова база клас.

→ move -> както при конструиране

• Консултация

Тема № 11:

- > статично св.- изборът на ф-я, която да се използва, се прави по време на време на конструиране. Избира се от членовете/дължимостта, от които се изброява.



-> динамично свързване - изборът на ф-я се прави по време на изпълнение на програмата (runtime), се прави посредством виртуални функции (членове, които могат да бъдат превзаписани)

-> overide - индицира, че дадената ф-я ще бъде превзаписана, предизвикан от други

-> final - никой надолу в наследството не може да превзаписва ф-та/да наследства дадения клас

-> виртуални таблици - клас, който не е абстрактен, има виртуална таблица, която преминава имена на дин. св.; членовете са същи в наследото; дава се достъпи, се правят г-дължимости и се ползват допълнителни методи; не се контира при к-к.

-> полиморфизъм - 1 ище, който има идентични

+> compile time -> function и operator overloading

+> runtime (дин. св.)

-> абстрактен клас - клас, който е предназначен за наследяване, не прави обекти от него; съдържа също абстрактни ф-и, като актите се различават от даден наследник, и той се прави абстрактен.

Тема №12:

- коллекция от обекти в полиморфна биерархия
 - > хеметогенен контейнер - клас, който поддържа коллекция от указатели към абстрактни клас
 - > копиране - чрез клониране (бр-я, в която ≠ обект пращи копие на себе си).

`Base* clone {
 return new A(*this); }`

-> транс - виртуален деструктор

-> конкретика - само ВСВ Factory function.

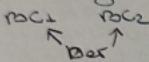
-> равновнаване - чрез глен-каст / dynamic cast

-> взаимно разпознаване:

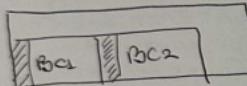
v. handle
handleA
handleB

```
f ( Base* lhs, Base* rhs ) {  
    lhs.handle(rhs);  
}  
A:: handle (Base* rhs) {  
    rhs->handleA(this); }
```

- Множествено наследяване -



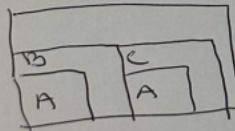
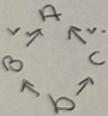
$BC1^* p1 = \& d$ > в този случаи $p1 \neq p2$
 $BC2^* p2 = \& d$



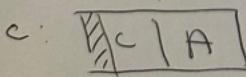
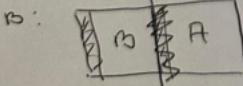
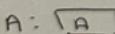
-> 2 базови класа
 => 2 вирт. таблицы

-> виртуалният таблица със собствен параметър Δ (активо отнасяне
 тръбва да направи указателя, за да намери адреса на Δ).
 $\Delta(BC1) = 0$
 $\Delta(BC2) = \text{sizeof}(BC2)$
 => к-ра и ~ викат крие и дескриптор на Base класовете

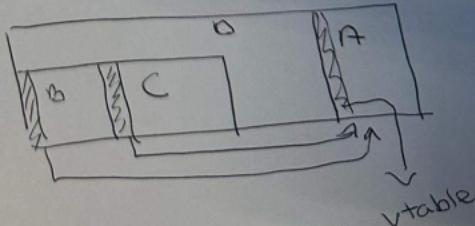
Динамични проблеми

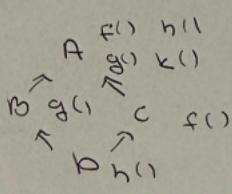


Ако искаме да имам дублиране на памет, използване виртуално наследяване. Тъкнулици на Вис също отговарят за използването на Δ .



D:





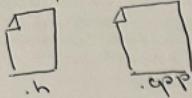
Б А синтаксич. табл:

$c :: f$	$-\Delta MOC$
$B :: g$	$-\Delta A$
$D :: h$	$-\Delta A$
$A :: k$	0

• Концептуализация

Тема № 13:

- шаблони - функции / класове / структури, с общи предназначение
- template <typename T> - параметричен по полиморфизъм



.h не винаги .cpp и не може да инициализира на .cpp да генерира функциите за дадени типове

=> или указваше в края на .cpp типовете, с които ще работиш или .hpp файл

-> шаблоните се запазват по време на компилиация

-> темпелейтични специализации - функции / клас, който се дели на различни за конкретен (vector<bool>)

-> наред, функции - ако някога се вика on =, то T трябва да има on.

-> примери:

-> vector, queue, stack

-> std::sort, std::find, std::find_if

-> типни указатели

• Типни указатели - обявяват клас около указателя, като дескриптор на указ. изтрява един. пакет.

-> unique_ptr - 1 указател за 1 обект; няма к-к и on =.

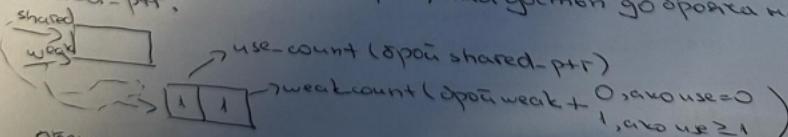
 +> make_unique<A>(3, 4, 5, 'A') - специална употреба на new

-> shared_ptr - и много указатели хоят обект, като пос. указател изтрява обекта

 +> make_shared<A>(...);

 +> чиста указател като ptr, който брои shared_ptr-ите на него + обект.

-> weak_ptr - not-owning ref. - насочване към обект, който е мениджиран от shared_ptr, не влияе на времето / създаването. Използване за да знаеш временно ходеше на място нещо; обикновено е промотиране от weak в shared, както и обратно да биде пренесено обратно изтрит, чието време до броята на shared_ptr.



Обектът е мяще, когато use-count=0
БРОЙ - когато weak-count=0.

Тема №3.

- fstream - потоки и файлы, и/o на уровне; последний на ifstream и ofstream

- > flush() - вспомогательная операция с текстом и выводом
 - > put/get - это вспомогательные функции
- введение в файлы:
 - struct A {
 - int x;
 - char a;
 - };
 - std::ofstream ofs("file.dat", std::ios::out | std::ios::binary);
 - ofs.write((const char*)&A, sizeof(A));
 - ofs.read((char*)&A, sizeof(A));
 - > замечание на обратку:
 - struct A {
 - int x;
 - char a;
 - };
 - std::ifstream ifs("file.dat", std::ios::in | std::ios::binary);
 - A obj;
 - ofs.write((const char*)&obj, sizeof(obj));
 - ofs.clear();
 - ofs.close();
- > чтение из обратки
 - std::ifstream ifs("file.dat", std::ios::in | std::ios::binary);
 - A obj;
 - ifs.read((char*)&obj, sizeof(obj));
- > замечание / чтение на массив от инкапсуляции:
 - struct Person {
 - char* name;
 - int age;
 - void readFromFile(std::ifstream& ifs) {
 - if (ifs.read((char*)&len, sizeof(len)) && len != 0);
 - ifs.read((char*)&len, sizeof(len));
 - name = new char [len];
 - ifs.read(name, len);
 - ifs.read((char*)&age, sizeof(int));
 - };
 - void saveToFile(std::ofstream& ofs) {
 - size_t len = strlen(name) + 1;
 - ofs.write((const char*)&len, sizeof(len));
 - ofs.write(name, len);
 - ofs.write((const char*)&age, sizeof(int));
 - };
- };
- int main() {
 - Person* people = new Person[100], size + n);
 - std::ifstream ifs("file.dat", std::ios::in | std::ios::binary);
 - ifs.read((char*)&n, sizeof(n));
 - Person* arr = new Person[n];
 - for (int i = 0; i < n; i++) {
 - arr[i].readFromFile(ifs);
 - };

-11 - замечание

-> замечание на глобальные объекты:

```
void replaceS(const char* fileName, char s, char toRep) {  
    std::fstream fs(fileName);  
    size_t cur = fs.tellg();  
    fs.seekg(0, std::ios::beg);  
    while (true) {  
        if (fs.get() == toRep)  
            fs.put(s);  
        else  
            fs.ignore();  
    }  
}
```