О C++

C++ включает в себя почти весь C. Если это не так - лучше не использовать такие расширения языка C.

Алгоритмы кэшей

LRU

Можно реализовать через массив, но тогда вытеснение O(n). Однако самое лучшее решение - двусвязный список для кэша и хэш таблица для проверки того, что данные закэшированы.
При реализации алгоритма на языке C тела структур не следует выносить в заголовочники. Тогда они будут доступны из разных модулей, что небезопасно. В заголовочнике должен в первую очередь находиться интерфейс, а не реализация. Во-первых это делает использование безопасным, а во-вторых, защищает от дублирования кода.

Вопросы, о которых нужно задумываться при написании алгоритмов

1. Маштабируемы ли алгоритмы?
2. Присутствуют ли собственные реализации? Собственных реализаций стандартных трукур данных нужно избегать.

```
Triangle::square() - метод square принадлежит пространству имен Triangle.
```

Базовые особенности C++

Обединение данных и методов к ним = классы C++

`this` ptr

Присутсвует неявно в качестве аргумента в каждом методе класса. Счиатется дурным тоном явно при обращении к полям класса писать `this->`. Его пишут тогда, когда хотят явно упомянуь о двухфазном разрешении имен.

```
#include <iostream>
#define $(x) std::cout << #x << " = " << x << std::endl
class A {
public:
    void print(int valA);
private:
    int valA{1};
    int valB{2};
};

void A::print(int valA) {
    $(valA);
    $(this->valA);
    $(valB);
    $(this->valB);
```

```
}

int main() {
    auto objA = A();
    objA.print(3);

}
```

Поля класса реализуются через указатели

Обобщение методов и шаблоны

Шаблоны лучше препроцессинга тем, что оно остается в рамках грамматики языка - дисциплина. Наглядный пример, чем шаблоны лучше макросов:

```
#define MAX(x, y) ( ( (x) > (y) ) ? (x) : (y) )
// тут могут возникнуть проблемы с побочными эффектами:
// MAX(a++, b--);
// MAX(2, "haha");
// могут возникнуть проблемы с производительностью:
// MAX(slow_foo(), 2); // slow_foo() вызовется аж дважды :(

template <typename T> T max(T x, T y) {
return (x > y)? x : y;
}
```

> Использование в C++ `void*` - очень дурной и небезопасный тон.
> **Нельзя использовать символьные имена с нижнего подчеркивания - они зарезервированы для нужд стандартной бибилиотеки.**

`splice` - метод стандартной библиотеки, позволяющий взять элемент (диапазон элементов) списка и перенести их в начало (к голове).

> Архаизм: объявлять и инициализировать переменные в начале функции упрощает вычисление размера стекового фрейма. Однако современные компиляторы этого правила не придерживаются. Вероятно, из-за назначения регистров.

```
// compile w/ -O1 -O2 -O3
#include <stdio.h>
int foo(int b) {
    int a = b + 4;
    printf("hello");
    return a;
}
```