Множественное наследование для исклчений

```
class ErrorSys{};
class ErrorNetwork{};
class ErrorInputSysNetwork: public ErrorSys, public ErrorNetwork{};

void f(){
    try{
        func_job_network(); // throw ErrorInputSysNetwork
    }
    catch(ErrorNetwork &e){
        throw;
    }
}
void g(){
    try{
        func_sys();
    }
    catch(ErrorSys &e){
        throw;
    }
}
int main(){
    try{
        f();
        g();
    }
    catch(ErrorInpytSysNetwork &e){
        ...
    }
}
```

Получается иерархия от частного к общему. Это может пригодиться, когда у нескольких разных ошибок одно и то же поведение

Потоки

Мультипрограммирование - способ организации вычислительного процесса, при котором на одном процессоре выполняется поочередно несколько программ.

Классификация параллельности по "зернистости" - какой размер атомарных операций, входящих в поток. Зернистость определяется по тому, сколько тактов уходит на атомарную операцию.

```
#include <thread>
#include <chrono>
#include <iostream>

long car[2] = {0, 0};
void car1(){
    while(1){
        if (car[0] > 99999999) car[0] = 0;
        else car[0]++;
    }
}
void car2(int start){
    car[1] = start;
    while(1){
        if (car[1] > 99999999) car[1] = 0;
```

```
        else car[1]++;
    }
}

int main(){
    std::thread f(car1);
    std::thread s(car2; 100);
    f.detach();
    s.detach();
    while(1){
        std::cout << car[0] << '-' << car[1] << std::endl;
        std::this_thread::sleep_for(std::chrono::second(1));
    }
}
```

простейший mutex описан в библиотеке

```
#include <thread>
#include <chrono>
#include <iostream>

long car = 0;
void car1(mutex &m){
    while(1){
        m.lock();
        if (car > 99999999) car = 0;
        else car++;
        m.unlock();
    }
}
void car2(mutex &m){
    while(1){
        m.lock();
        if (car <= 0) car = 999999999;
        else car--;
        m.unlock();
    }
}

int main(){
    std::mutex m;
    std::thread f(car, m);
    std::thread s(car, m);
    f.detach();
    s.detach();
    while(1){
        std::cout << car[0] << '-' << car[1] << std::endl;
        std::this_thread::sleep_for(std::chrono::second(1));
    }
}
```

Если вместо detach выполнить join, то вывод в main не выполнится никогда. Чтобы добиться вывода, нужно создать еще один поток и вызывать его. Чтобы завершить все потоки, можно вызвать std::terminate. У каждого потока можно вызвать getid. Можно вызвать yield, чтобы приостановить выполнение потока до следующего входа в цикл. Чтобы передать в std::thread обект, нужно передать ему объект с перегруженным оператором-функтором.

```
class Run{
    void operator()(){
        ...
    }
}
```

С помощью std::ref можно передавать указатель на переменную

```
if (i > f()){
    st1;
}else{
    st2;
}
```

Шаблоны типа

```
class char_stack{
    char *v;
    char *p;
    int sz;
    public:
    char_stack(){};
    char_stack(int size){
        p = v = new char[sz=size];
    }
    void push(char *c){
        *p++ = c;
    }
    char pop(){
        return *--p;
    }
    int getSize(){
        return p-v;
    }
}
```

```
template <class T>;
class stack{
    T *v;
    T *p;
    int sz;
    public:
    stack(){};
    stack(int size){
        p = v = new T[sz=size];
    }
    void push(T *c){
        *p++ = c;
    }
    T pop(){
        return *--p;
    }
    int getSize(){
        return p-v;
    }
}
```

В описании template можно использовать typename (старый стандарт) или class (новый)

Отношение инстанцирования

```
stack <A> a;
stack <A*> a;
stack <int> a;

template <T, a>
stack <int, 10>;
```