1. Константы

```
namespace
{
constexpr int kMaxAmountOfNuggets = 10'000;
constexpr int kDaysToWork = 5;
} // namespace
```

kMaxAmountOfNuggets – то сколько максимально можно скушать наггетсов

kDaysToWork – сколько дней проработает Кук перед тем как уволиться

2. Глобальные переменные

```
std::mutex m;
int fat_men_eaten_count = 0;
```

m — мьютекс для блокировки потоков. Fat_men_eaten_count — счетчик для учета того какое количество толстяков поели.

```
enum class ProgramState : uint8_t
{
  Running,
  CookFired,
  CookQuit,
  CookNoSalary,
  Finished
};
ProgramState program_state = ProgramState::Running;
```

Program_state – для того чтобы отслеживать в каком сейчас состоянии программа (Запущена, Кук уволен, Кук без зп, etc)

```
enum class Turn : uint8_t
{
  Cook,
  FatMen
};
Turn current_turn = Turn::Cook;
```

Current_turn — для того чтобы определить чей сейчас ход.

3. Класс толстяка

```
class FatMan
{
public:
  FatMan() = default;
  FatMan(int const gluttony): gluttony_(gluttony) {}

  void Eat(int & dish)
  {
    while (true)
    {
      m.lock();
      if (program_state != ProgramState::Running || IsBlown())
      {
        m.unlock();
        break;
      }

      if (current_turn != Turn::FatMen || has_eaten_this_round_)
      {
        m.unlock();
        std::this_thread::yield();
        continue;
      }
```

```
    if (dish ≥ gluttony_)
    {
      amount_of_nuggets_ += gluttony_;
      dish -= gluttony_;
      has_eaten_this_round_ = true;
      fat_men_eaten_count += 1;

      if (IsBlown())
      {
        m.unlock();
        break;
      }
    }
    else
    {
      amount_of_nuggets_ += dish;
      dish -= gluttony_;
      program_state = ProgramState::CookFired;
      has_eaten_this_round_ = true;
      fat_men_eaten_count += 1;
    }

    m.unlock();
    std::this_thread::yield();

    if (program_state != ProgramState::Running)
    {
      break;
    }
  }
}

  bool IsBlown() const { return amount_of_nuggets_ > kMaxAmountOfNuggets; }
  int GetEaten() const { return amount_of_nuggets_; }
  void ResetRound() { has_eaten_this_round_ = false; }

private:
  int amount_of_nuggets_ = 0;
  int gluttony_ = 0;
  bool has_eaten_this_round_ = false;
};
```

Поля: amount_of_nuggets – количество съеденных наггетсов. Gluttony – то сколько может съесть за раз. Has_eaten_this_round – ел ли в этом раунде.

Методы:

IsBlown() – возвращает “взорвался” ли объект. GetEaten() – геттер для amount_of_nuggets. ResetRound() – выставляет has_eaten_this_round в false.

Eat() – метод для того чтобы толстяк ел. Работает следующим образом: Зацикливаемся с помощью while(true), дальше получаем блокировку с помощью m.lock(). После чего следует проверка на то что программа еще находится в рабочем состоянии, а толстяк еще не взорвался.

Дальше идёт проверка на то чей сейчас ход и ел ли объект уже.

Ну и наконец проверка на то что находится ли в тарелке больше чем может съесть толстяк. Если это так, то толстяк съедает то что может из тарелки, после чего следует проверка на то что взорвался толстяк или нет.

Если же в тарелке меньше чем может съесть, то он съедает всё что в тарелке и Кука увольняют.

4. Класс Кука

```
class Cook
{
public:
  Cook() = default;
  Cook(int const efficiency_factor) : efficiency_factor_(efficiency_factor) {}

  void Serve(int & dish1, int & dish2, int & dish3)
  {
    auto start = std::chrono::high_resolution_clock::now();
    while (program_state == ProgramState::Running)
    {
      m.lock();

      if (current_turn != Turn::Cook) {
        // std::println("Non cook turn");
        // std::println("{}", Print(program_state));
        m.unlock();
        std::this_thread::yield();
        continue;
      }
```

```
    auto current_time = std::chrono::high_resolution_clock::now();
    auto elapsed = std::chrono::duration_cast<std::chrono::seconds>(current_time - start).count();
    if (elapsed >= kDaysToWork)
    {
      program_state = ProgramState::CookQuit;
      m.unlock();
      break;
    }

    // std::println("Serve");
    dish1 += efficiency_factor_;
    dish2 += efficiency_factor_;
    dish3 += efficiency_factor_;

    current_turn = Turn::FatMen;
    m.unlock();

    std::this_thread::yield();
    if (program_state != ProgramState::Running) {
      break;
    }
  }
}
```

```
private:
  int efficiency_factor_ = 0;
};
```

Поля: efficiency_factor – то сколько Кук может наложить в тарелку наггетсов.

Методы: Serve(int, int, int) – метод для того чтобы накладывать наггетсы в тарелки. Работает следующим образом: засекается старт работы Кука. После чего мы зацикливаемся пока программа находится в работающем состоянии. После чего получаем блокировку.

Дальше проверяем, является ли настоящий ход ходом Кука. Дальше мы сравниваем количество проработанных дней Куком с количеством дней когда пора уволиться. Если больше или равно, то Кук уходит сам.

После проверки Кук накладывает в каждую тарелку по несколько наггетсов. И передаёт ход дальше толстякам.

5. Точка входа

```cpp
int main()
{
  RunScenario("CookQuit", 50, 50, 50, 100);
  RunScenario("CookNoSalary", 1000, 1000, 1000, 1000);
  RunScenario("CookIsFired", 50, 50, 5000, 1);

  return 0;
}
```

Запуск 3 сценариев.

6. Функция запуска сценариев.

```cpp
void RunScenario(std::string const & scenario_name, int gluttony1, int gluttony2, int gluttony3, int efficiency_factor)
{
  std::println("\n=== {} ===", scenario_name);
  std::println("gluttony1: {}, gluttony2: {}, gluttony3: {} efficiency_factor {}", gluttony1, gluttony2, gluttony3, efficiency_factor);
  int dish1 = 3000;
  int dish2 = 3000;
  int dish3 = 3000;

  program_state = ProgramState::Running;
  current_turn = Turn::Cook;

  FatMan fat_man1(gluttony1);
  FatMan fat_man2(gluttony2);
  FatMan fat_man3(gluttony3);

  Cook cook(efficiency_factor);

  std::thread fat_man1_t(&FatMan::Eat, &fat_man1, std::ref(dish1));
  std::thread fat_man2_t(&FatMan::Eat, &fat_man2, std::ref(dish2));
  std::thread fat_man3_t(&FatMan::Eat, &fat_man3, std::ref(dish3));
  std::thread cook_t(&Cook::Serve, &cook, std::ref(dish1), std::ref(dish2), std::ref(dish3));

  ProgramState current_state = ProgramState::Running;
  while (current_state == ProgramState::Running)
  {
    std::this_thread::sleep_for(std::chrono::milliseconds(100));
    current_state = CheckState(fat_man1, fat_man2, fat_man3, dish1, dish2, dish3);
  }

  fat_man1_t.join();
  fat_man2_t.join();
  fat_man3_t.join();
  cook_t.join();

  std::println("dish1: {}, dish2: {}, dish3: {}", dish1, dish2, dish3);
  std::println("Eaten:");
  std::println("fat_man1: {}, fat_man2: {}, fat_man3: {}", fat_man1.GetEaten(), fat_man2.GetEaten(), fat_man3.GetEaten());

  std::println("{}", ToString(current_state));
}
```

Функция аккумулирует в себе инициализацию необходимых объектов классов и создание потоков, передачу данных в потоки. Раз в 100 миллисекунд вызывает проверку состояния программы. А так же отвечает за вывод.

7. Проверка состояния программы.

```
ProgramState CheckState(
  FatMan & fat_man1, FatMan & fat_man2, FatMan & fat_man3,
  int dish1, int dish2, int dish3)
{
  m.lock();
  if (program_state != ProgramState::Running) {
    m.unlock();
    return program_state;
  }

  if (fat_man1.IsBlown() && fat_man2.IsBlown() && fat_man3.IsBlown())
  {
    program_state = ProgramState::CookNoSalary;
    current_turn = Turn::Cook;
  }
  else if (dish1 < 0 || dish2 < 0 || dish3 < 0)
  {
    program_state = ProgramState::CookFired;
    current_turn = Turn::Cook;
  }
  else if (current_turn == Turn::FatMen && fat_men_eaten_count >= 3)
  {
    current_turn = Turn::Cook;
    fat_men_eaten_count = 0;
    fat_man1.ResetRound();
    fat_man2.ResetRound();
    fat_man3.ResetRound();
  }

  ProgramState current_state = program_state;
  m.unlock();
  return current_state;
}
```

Берём блокировку. Первая проверка на то что сейчас работающее состояние. Если нет, то разблокируем поток и вернём состояние.

Дальше проверяем, если все толстяки взорвались, то выставляем соответствующее состояние и передаем ход Куку

Следующая проверка это на то что в тарелках может не хватать наггетсов. Тогда кука уволят.

И проверка на то чей сейчас ход. Отслеживается чей сейчас ход и сколько уже толстяков поели. В случае истины передаем ход куку и обнуляем переменные.

В итоге возвращаем настоящее состояние программы.

8. Вывод

```
=== CookQuit ===
gluttony1: 50, gluttony2: 50, gluttony3: 50 efficiency_factor 100
dish1: 5500, dish2: 5500, dish3: 5500
Eaten:
fat_man1: 2500, fat_man2: 2500, fat_man3: 2500
CookQuit

=== CookNoSalary ===
gluttony1: 1000, gluttony2: 1000, gluttony3: 1000 efficiency_factor 1000
dish1: 3000, dish2: 3000, dish3: 3000
Eaten:
fat_man1: 11000, fat_man2: 11000, fat_man3: 11000
CookNoSalary

=== CookIsFired ===
gluttony1: 50, gluttony2: 50, gluttony3: 5000 efficiency_factor 1
dish1: 2951, dish2: 2951, dish3: -1999
Eaten:
fat_man1: 50, fat_man2: 50, fat_man3: 3001
CookFired
```