Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3213

Поленов К.А.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2025

Задача №A «Агроном-любитель»

```cpp
#include <cstdint>
#include <iostream>

int main() {
  uint64_t n;
  std::cin >> n;

  auto* flowers = new int64_t[n];

  for (size_t i = 0; i < n; i++) {
    int64_t flower;
    std::cin >> flower;

    flowers[i] = flower;
  }

  int64_t current, previous = -1, past_previous = -2;
  uint64_t begin_m = 1, end_m = 1, max = 0, begin = 1, end, delta = 0;

  for (uint64_t i = 0; i < n; i++) {
    current = flowers[i];
    end = i + 1;

    if (i >= 2 && current == previous && previous == past_previous) {
      end--;
      delta = end - begin;
      begin = end;
    } else {
      delta = end - begin;
    }

    if (delta > max) {
      max = delta;
      begin_m = begin;
      end_m = end;
    }

    past_previous = previous;
    previous = current;
  }

  std::cout << begin_m << " " << end_m;
  delete[] flowers;
  return 0;
}

```

Пояснение к примененному алгоритму:

На вход подаётся n. Считывается n чисел, каждая из которых характеризует тип цветка. Алгоритм заключается в обходе массива чисел, используя 3 переменные: текущий цветок, предыдущий и предпредыдущий. Когда алгоритм проход 3й цветок, все переменные цветков заполнены корректными значениями. Если все три числа равны, уменьшаем крайний индекс позиции, так как мы его увеличили выше, и назначаем конец текущего отрезка началом следующего, затем вычисляем расстояние между началом и концом отрезка и переопределяем его и начало и конец отрезка, если расстояние больше максимума. Затем обновляем переменные предыдущего и предпредущего цветков.

Задача №B «Зоопарк Глеба»

```cpp
#include <iostream>
#include <map>
#include <stack>

bool isTrapped(char animal, char trap) {
  return animal != trap && toupper(animal) == toupper(trap);
}

int main() {
  std::string line;
  std::cin >> line;

  std::string chars;
  std::stack<unsigned int> traps_indexes;
  std::stack<unsigned int> animals_indexes;
  unsigned int trap_index = 0;
  unsigned int animal_index = 0;
  std::map<unsigned int, unsigned int> ans_indexes;

  for (size_t i = 0; i < line.length(); i++) {
    if (isupper(line[i])) {
      trap_index++;
      traps_indexes.push(trap_index);
    } else {
      animal_index++;
      animals_indexes.push(animal_index);
    }

    if (line[i] == '\n')
      break;

    if (!chars.empty() && isTrapped(line[i], chars.back())) {
      ans_indexes[traps_indexes.top()] = animals_indexes.top();

      animals_indexes.pop();
      traps_indexes.pop();
      chars.pop_back();
    } else {
      chars += line[i];
    }
  }

  if (!chars.empty())
    std::cout << "Impossible\n";
  else {
    std::cout << "Possible" << std::endl;
    for (const auto& [key, val] : ans_indexes) {
      std::cout << val << " ";
    }
  }

  return 0;
}
```

Пояснение к примененному алгоритму:

Используется два стека для сохранения и отслеживания индексов ловушек и животных, а также строка, которая заполняется текущими символами ловушек и животных без пар. Для сохранения результатов используется словарь <индекс ловушки, индекс животного>.

Алгоритм проходит по введенной строке. Если символ заглавный, увеличивает индекс ловушки/животного и кладет на соответствующий стек. Если строка-аккумулятор не пустая и крайние символы в строке и ловушке отличаются лишь регистром, со стека снимаются и кладутся в словарь индексы пары, а также из строки

убирается символ, которому нашлась пара. Иначе к строке прибавляется символ, которому пары пока еще не нашлось.

Если после прохода по строке строка-аккумулятор содержит хотя бы один символ, то есть символ/символы которому/которым не нашлось пары, то это означает, что не нашлось пары ловушке или зверю, то есть существует пересечение.

Задача №C «Конфигурационный файл»

```cpp
#include <stack>
#include <cstring>
#include <map>
#include <vector>
std::pair<std::string, std::string> parse_line(const std::string &line) {
    size_t equality_char_ind = line.find( c: '=');
    auto var_name : string  = line.substr( pos: 0,  n: equality_char_ind);
    auto var_value : string  = line.substr( pos: equality_char_ind + 1,  n: line.length());

    return std::make_pair( &: var_name,  &: var_value);
}

int get_var_value(std::map<std::string, std::stack<int>> &map, const std::string &name) {
    if (!map.contains( x: name) || map[name].empty())
        return 0;
    return map[name].top();
}

void clear_scope_vars(std::map<std::string, std::stack<int>> &map, std::stack<std::vector<std::string>> &scopes) {
    auto scope : vector<string>  = scopes.top();
    for (size_t i = 0; i < scope.size(); i++) {
        map[scope[i]].pop();
    }
    scopes.pop();
}

bool is_digit(std::string &token) {
    return isdigit( c: token[0]) || token[0] == '-';
}

int main() {
    // словарь <имя, стек значений переменной>
    std::map<std::string, std::stack<int>> vars_pairs;
    // стек скоупов. Каждый скоуп - вектор из имен переменных скоупа
    std::stack<std::vector<std::string>> scopes;
    scopes.emplace();

    std::string line;
    while (std::cin >> line) {

        if (line == "{")
            scopes.emplace();
        else if (line == "}")
            clear_scope_vars( &: vars_pairs,  &: scopes);
        else {
            auto args : pair<string, string>  = parse_line(line);

            if (is_digit( &: args.second))
                vars_pairs[args.first].push( x: stoi( str: args.second));
            else {
                int value = get_var_value( &: vars_pairs,  name: args.second);
                vars_pairs[args.first].push( x: value);
                std::cout << value << std::endl;
            }
            scopes.top().push_back(args.first);

        }
    }
    return 0;
}
```

Пояснение к примененному алгоритму:

Используется словарь <имя переменной, стек её значений> и стек векторов из строк, где каждый вектор представляет собой область видимости текущего уровня, а строки в нем – переменное этого уровня области видимости.

Перед проходом по строке, предварительно инициализируем пустой вектор, представляющий собой глобальную область видимости. Алгоритм проходит по строке. Если встретился символ «{», инициализируется новый вектор области видимости в стеке областей, если встретился символ «}», значения с вершин стеков всех переменных текущей области видимости очищаются, затем очищается сама область видимости. Если ни один из символов фигурных скобок не был встречен, строчка разделяется на аргументы. Если второй аргумент число, то на стек значений переменной (первого аргумента) кладется это число. Если это имя другой переменной, из словаря достается значение с вершины стека значений второй переменной и кладется на стек значений первой переменной. Значение второй переменной печатается в консоль. В вектор с вершины стека областей видимости добавляется новая переменная.

Задача №D «Профессор Хаос»

```cpp
#include <iostream>

int main() {
  int a = 1, b = 1, remaining_a = 0;
  short c = 0, d = 1;
  unsigned long long k = 1;

  std::cin >> a >> b >> c >> d >> k;

  while (k > 0) {
    a = a * b - c;

    if (a <= 0) {
      a = 0;
      break;
    }
    if (a > d)
      a = d;

    if (a == remaining_a)
      break;
    remaining_a = a;

    k--;
  }

  std::cout << a;
  return 0;
}
```

Пояснение к примененному алгоритму:

На вход подаются значения a, b, c, d, k, где a – начальное количество бактерий в контейнере, b – количество новых бактерий после деления одной бактерии, c – количество бактерий для эксперимента, d – вместимость контейнера, k – количество дней.

Пока дни не закончились, алгоритм выполняет следующие действия. Новое количество бактерий – это размноженные бактерии минус бактерии, взятые на опыты, если количество новых бактерий <= 0, то есть меньше или равно c, то все бактерии закончились и эксперимент считается завершенным. Если бактерий больше d, то новое количество бактерий равно d, тк контейнер не может вместить больше. Так как существует вариант, при котором после очередного размножения бактерий и утилизации части из них на опыты, a остается таким же, каким и было. Если такое произошло, то все оставшиеся дни не дадут новых результатов, и поэтому в целях повышения скорости программы, можно досрочно выйти из цикла. Если такой ситуации нет, сохраняем оставшиеся бактерии в специальную переменную для отслеживания подобных случаев.