**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕНЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.Шухова»**

**(БГТУ им. В.Г.Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа № 6

По дисциплине: Алгоритмы и СД

Тема: **Структуры данных «стек» и «очередь» (Pascal/С)**

Выполнил: ст.группы ПВ - 21

Браткова И.О.

Белгород 2017

**Цель работы:** изучить СД типа «стек» и «очередь», научиться их программно реализовывать и использовать.

З а д а н и е

1. Для СД типа «стек» и «очередь» определить:

Абстрактный уровень представления СД:

1.1.1. Характер организованности и изменчивости.

1.1.2. Набор допустимых операций.

1.2. Физический уровень представления СД:

1.2.1. Схему хранения.

1.2.2. Объем памяти, занимаемый экземпляром СД.

1.2.3. Формат внутреннего представления СД и способ его ин-терпретации.

1.2.4. Характеристику допустимых значений.

1.2.5. Тип доступа к элементам

Логический уровень представления СД.

Способ описания СД и экземпляра СД на языке программирования.

2. Реализовать СД типа «стек» и «очередь» в соответствии с вариантом индивидуального задания в виде модуля.

3. Разработать программу, моделирующую вычислительную систему с постоянным шагом по времени (дискретное время) в соответствии с вариантом индивидуального задания (табл.16) с использованием модуля, полученного в результате выполнения пункта 2. Результат работы программы представить в виде таблицы 15. В первом столбце указывается время моделирования 0, 1, 2, …, N. Во втором — для каждого момента времени указываются имена объектов (очереди — F1, F2, …, FN; стеки — S1, S2, …, SM; процессоры — P1, P2, …, PK), а в третьем — задачи (имя, время), находящиеся в объектах.

***Задание 1***

1. Абстрактный уровень представления СД:
   1. Характер организованности и изменчивости.

Стек: Линейная структура, последовательность.

Очередь: Линейная структура, последовательность.

* 1. Набор допустимых операций

Стек: Инициализация. Включение элемента. Исключение элемента. Чтение элемента. Проверка пустоты стека. Уничтожение стека.

Очередь: Инициализация. Включение элемента. Исключение элемента. Проверка пустоты очереди. Уничтожение очереди.

1. Физический уровень представления СД:
   1. Схему хранения.

Стек: Последовательная или связная.

Очередь: Последовательная или связная.

* 1. Объем памяти, занимаемый экземпляром СД.

Стек: Зависит от реализации.

Очередь: Зависит от реализации.

* 1. Формат внутреннего представления СД и способ его интерпретации.

Стек: Может быть реализован в динамической/статической памяти. ПЛС, ОЛС.

Очередь: Может быть реализован в динамической/статической памяти. ПЛС, ОЛС.

* 1. Характеристику допустимых значений.

Стек: Зависит от реализации.

Очередь: Зависит от реализации.

* 1. Тип доступа к элементам.

Стек: Последовательный.

Очередь: Последовательный.

1. Логический уровень представления СД.

Стек: Stack SS;

Очередь: FIFO FF;

Система состоит из двух процессоров P1 и P2, трех очередей F0, F1, F2 и двух стеков S1 и S2.

О

Ч

Е

Р

Е

Д

Ь

F1

О

Ч

Е

Р

Е

Д

Ь

F2

P1

Стек S1

P2

Стек S2

Генератор задач

О

Ч

Е

Р

Е

Д

Ь

F0

***Задание 2:***

Был создан заголовочный файл, содержащий в себе структуру запроса:

typedef struct TInquiry

{

char Name[10]; // имя запроса

unsigned Time1; // время обслуживания

// unsigned Time2; // время обслуживания

unsigned short int P; // приоритет 0 — высший, 1 — средний, 2 - нисший

} TInquiry;

**Стек: stack.h**

#include <stdio.h> #include “Zapros.h” #include <stdlib.h>

#define SIZE\_STACK\_ARRAY 100

/\*Описание исключительных ситуаций\*/

const int okStackArray = 0; // Все нормально

const int fullStackArray = 1; // Стек переполнен

const int emptyStackArray = 2; // Стек пуст

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

extern int errorStackArray; //Переменная ошибок

typedef TInquiry stackArrayBaseType; //Базовый тип стека

/\*Дескриптор стека\*/

typedef struct {

stackArrayBaseType buf[SIZE\_STACK\_ARRAY]; // Буфер стека

unsigned uk; // Указатель

} StackArray;

void initStackArray(StackArray \*S);

void putStackArray(StackArray \*S, stackArrayBaseType E);

void getStackArray(StackArray \*S, stackArrayBaseType \*E);

int isFullStackArray(StackArray \*S);

int isEmptyStackArray(StackArray \*S);

void initStackArray(StackArray \*S) /\*Инициализация стека\*/

{ S->uk = 0; errorStackArray = okStackArray; }

void putStackArray(StackArray \*S, stackArrayBaseType E) /\*Включение в стек\*/

{

if (isFullStackArray(S)) return;

S->buf[S->uk] = E; S->uk++;

}

void getStackArray(StackArray \*S, stackArrayBaseType \*E) /\*Исключение\*/

{

if (isEmptyStackArray(S)) return;

\*E = S->buf[S->uk - 1]; S->uk--;

}

int isFullStackArray(StackArray \*S) /\*Предикат: полон ли стек\*/

{

if (S->uk == SIZE\_STACK\_ARRAY)

{ errorStackArray = fullStackArray; return 1; }

return 0;

}

int isEmptyStackArray(StackArray \*S) /\*Предикат: пуст ли стек\*/

{if (S->uk == 0) { errorStackArray = emptyStackArray; return 1; }return 0; }

**Очередь: Queue.h**

#include <stdio.h> #include “Zapros.h” #include <stdlib.h>

#define SIZE\_QUEUE\_ARRAY 3

/\*Описание исключительных ситуаций\*/

const int okQueueArray = 0; // Все нормально

const int fullQueueArray = 1; // Очередь переполнена

const int emptyQueueArray = 2; // Очередь пуста

extern int errorQueueArray;

typedef TInquiry queueArrayBaseType; /\*Базовый тип очереди\*/

typedef struct

{

queueArrayBaseType buf[SIZE\_QUEUE\_ARRAY]; // Буфер очереди

unsigned ukEnd; // Указатель на хвост (по нему включают)

unsigned ukBegin; // Указатель на голову (по нему исключают)

unsigned len; // Количество элементов в очереди

} QueueArray;

int errorQueueArray;

void initQueueArray(QueueArray \*F);

void putQueueArray(QueueArray \*F, queueArrayBaseType E);

void getQueueArray(QueueArray \*F, queueArrayBaseType \*E);

int isFullQueueArray(QueueArray \*F);

int isEmptyQueueArray(QueueArray \*F);

void initQueueArray(QueueArray \*F) //Инициализация очереди

{ F->ukBegin = 0; F->ukEnd = 0; F->len = 0; errorQueueArray = okQueueArray;}

void putQueueArray(QueueArray \*F, queueArrayBaseType E) /\*Включение в очередь\*/

{

if (isFullQueueArray(F)) return;

F->buf[F->ukEnd] = E; // Включение элемента

F->ukEnd = (F->ukEnd + 1) % SIZE\_QUEUE\_ARRAY; // Сдвиг указателя

F->len++; // Увеличение количества элементов очереди

}

void getQueueArray(QueueArray \*F, queueArrayBaseType \*E) /\*Исключение из очереди\*/

{

if (isEmptyQueueArray(F)) return;

\*E = F->buf[F->ukBegin]; // Запись элемента в переменную

F->ukBegin = (F->ukBegin + 1) % SIZE\_QUEUE\_ARRAY; // Сдвиг указателя

F->len--; // Уменьшение длины

}

int isFullQueueArray(QueueArray \*F)

{if (F->len == SIZE\_QUEUE\_ARRAY){errorQueueArray = fullQueueArray;return 1;}

return 0;

}

int isEmptyQueueArray(QueueArray \*F) /\*Предикат: пуста ли очередь\*/

{

if (F->len == 0)

{

errorQueueArray = emptyQueueArray;

return 1;

}

return 0;

}

**Функции работы с запросом:**

void WriteOneZapr (TInquiry a) //Вывод одного запроса

{

printf("name : %s\n", a.Name);

printf("time : %d\n", a.Time1);

printf("prior : %d\n", a.P);

}

void ReadOneZapr (TInquiry a) //Считывание одного запроса

{ scanf("%s", &a.Name); scanf("%d", &a.Time1); scanf("%d", &a.P); }

void ReadZapros (TInquiry \*a, size\_t n) //Считывание массива запросов длины n

{

int i;

for (i=0; i<n; i++)

{

printf("name >> "); scanf("%s", &a[i].Name);

printf("time1 >> "); scanf("%d", &a[i].Time1);

printf("prior >> "); scanf("%d", &a[i].P);

}

}

void WriteZapros (TInquiry\* a, size\_t n) //Вывод массива запросов длины n

{

int i;

for (i=0; i<n; i++)

{

printf("name : %s\n", a[i].Name);

printf("time : %d\n", a[i].Time1);

printf("prior : %d\n", a[i].P);

}

}

void GenZapros (TInquiry \*a) //Генерация одного запроса

{

int i;

for (i=0; i<10; i++)

a->Name[i] = rand()%(90-65+1) + 65;

a->P = rand()%(2-0+1) + 0;

a->Time1 = rand()%(10-2+1) + 2;

}

void GenMassZapros (TInquiry \*a, size\_t n) //Генерация массива запросов

{

int i;

for (i=0; i<n; i++)

GenZapros(&a[i]);

}

void MassFifoRaspr (TInquiry \*a, QueueArray \*F0, QueueArray \*F1, QueueArray \*F2, size\_t n)

//Распределение массива запросов длины n по очередям

{

int i;

for (i=0; i<n; i++)

{

if (a[i].P == 0) putQueueArray(F0, a[i]);

if (a[i].P == 1) putQueueArray(F1, a[i]);

if (a[i].P == 2) putQueueArray(F2, a[i]);

}

}

void FifoRaspr (TInquiry \*a, QueueArray \*F0, QueueArray \*F1, QueueArray \*F2)

{

if (a.P == 0) putQueueArray(&F0, a);

if (a.P == 1) putQueueArray(&F1, a);

if (a.P == 2) putQueueArray(&F2, a);

}

Основная программа 1 (генерация перед выполнением)

int main()

{

srand(time(NULL));

int n=3, i=0; printf("n >> "); //количество запросов

scanf("%d", &n);

TInquiry P1, P2, tmp0, tmp00, tmp1, tmp2, workz;

TInquiry \*a = (TInquiry\*)calloc(n, sizeof(TInquiry)); //выделение памяти на массив

GenMassZapros(a, n); //генерация массива запросов

WriteZapros(a, n);

QueueArray F0, F1, F2; //Объявление и инициализация очередей

initQueueArray(&F0);

initQueueArray(&F1);

initQueueArray(&F2);

StackArray S1, S2; //Объявление и инициализация стеков

initStackArray(&S1);

initStackArray(&S2);

MassFifoRaspr(a, &F1, &F2, &F0, n); //распределили запросы по очередям

while (isEmptyStackArray(&S1) || isEmptyStackArray(&S2) || !isEmptyQueueArray(&F0) || !isEmptyQueueArray(&F1) || !isEmptyQueueArray(&F2) || P1.Time1!=0 || P2.Time1!=0)

{

if (!isEmptyQueueArray(&F0)) //если нулевая очередь не пуста

{

if (P1.Time1==0 && P2.Time1 == 0) //процессоры пусты

P1=tmp0;

if (P1.Time1!=0 && P2.Time1==0) //пуст второй

P2=tmp0;

if (P1.Time1==0 && P2.Time1!=0) //пуст первый, не пуст второй

P1=tmp0;

if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P1.P)

{

putStackArray(&S1, P1); //поместили в стек то что было

P1=tmp0; //записали новый процесс

}

else if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P2.P)

{

putStackArray(&S2, P2); //поместили в стек то что было

P2=tmp0; //записали новый процесс

}

}

else

{

if (P1.Time1!=0) P1.Time1--; //уменьшаем время в процессоре 1

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S1)) getStackArray(&S1, &P1);

else if (!isEmptyQueueArray(&F0)) getQueueArray(&F0, &P1);

}

if (P2.Time1!=0) P2.Time1--; //уменьшаем время в процессоре 1

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S2)) getStackArray(&S2, &P2);

else if (!isEmptyQueueArray(&F0)) getQueueArray(&F0, &tmp0);

}

}

if (!isEmptyQueueArray(&F1))

{

if (P1.Time1==0 && P2.Time1 == 0) //процессоры пусты

P1=tmp0;

if (P1.Time1!=0 && P2.Time1==0) //пуст второй

P2=tmp0;

if (P1.Time1==0 && P2.Time1!=0) //пуст первый, не пуст второй

P1=tmp0;

if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P1.P)

{

putStackArray(&S1, P1); //поместили в стек то что было

P1=tmp0; //записали новый процесс

}

else if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P2.P)

{

putStackArray(&S2, P2); //поместили в стек то что было

P2=tmp0; //записали новый процесс

}

}

else

{

if (P1.Time1!=0) P1.Time1--; //уменьшаем время в процессоре 1

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S1)) getStackArray(&S1, &P1);

else if (!isEmptyQueueArray(&F1)) getQueueArray(&F1, &tmp0);

}

if (P2.Time1!=0) P2.Time1--; //уменьшаем время в процессоре 1

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S2)) getStackArray(&S2, &P2);

else if (!isEmptyQueueArray(&F1)) getQueueArray(&F1, &tmp0);

}

}

if (!isEmptyQueueArray(&F2))

{

if (P1.Time1==0 && P2.Time1 == 0) //процессоры пусты

P1=tmp0;

if (P1.Time1!=0 && P2.Time1==0) //пуст второй

P2=tmp0;

if (P1.Time1==0 && P2.Time1!=0) //пуст первый, не пуст второй

P1=tmp0;

if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P1.P)

{

putStackArray(&S1, P1); //поместили в стек то что было

P1=tmp0; //записали новый процесс

}

else if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P2.P)

{

putStackArray(&S2, P2); //поместили в стек то что было

P2=tmp0; //записали новый процесс

}

}

else

{

if (P1.Time1!=0) P1.Time1--; //уменьшаем время в процессоре 1

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S1)) getStackArray(&S1, &P1);

else if (!isEmptyQueueArray(&F2)) getQueueArray(&F2, &tmp0);

}

if (P2.Time1!=0) P2.Time1--; //уменьшаем время в процессоре 1

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S2))

getStackArray(&S2, &P2);

else if (!isEmptyQueueArray(&F2))

getQueueArray(&F2, &tmp0);

}

}

}

return 0;

}

Основная программа 2 (генерация во время выполнения)

int main()

{

srand(time(NULL)); int n, i=0; printf("n >> "); scanf("%d", &n);

TInquiry P1, P2, tmp0; QueueArray F0, F1, F2;

initQueueArray(&F0); initQueueArray(&F1); initQueueArray(&F2);

StackArray S1, S2; initStackArray(&S1); initStackArray(&S2);

GenZapros(&tmp0); i=1;

while (i>n || isEmptyStackArray(&S1) || isEmptyStackArray(&S2) || !isEmptyQueueArray(&F0) || !isEmptyQueueArray(&F1) || !isEmptyQueueArray(&F2) || P1.Time1!=0 || P2.Time1!=0)

{

if (!isEmptyQueueArray(&F0)) //если нулевая не пуста

{

if (P1.Time1==0 && P2.Time1 == 0) //процессоры пусты

P1=tmp0;

if (P1.Time1!=0 && P2.Time1==0) //пуст второй

P2=tmp0;

if (P1.Time1==0 && P2.Time1!=0) //пуст первый, не пуст второй

P1=tmp0;

if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P1.P)

{

putStackArray(&S1, P1); //поместили в стек то что было

P1=tmp0; //записали новый процесс

}

else if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P2.P)

{

putStackArray(&S2, P2); //поместили в стек то что было

P2=tmp0; //записали новый процесс

}

}

else

{

if (P1.Time1!=0) P1.Time1--;

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S1)) getStackArray(&S1, &P1);

else if (!isEmptyQueueArray(&F0)) getQueueArray(&F0, &P1);

}

if (P2.Time1!=0) P2.Time1--;

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S2)) getStackArray(&S2, &P2);

else if (!isEmptyQueueArray(&F0)) getQueueArray(&F0, &tmp0);

}

}

if (!isEmptyQueueArray(&F1)) //если нулевая не пуста

{

//getQueueArray(&F0, &tmp0);

if (P1.Time1==0 && P2.Time1 == 0) //процессоры пусты

P1=tmp0;

if (P1.Time1!=0 && P2.Time1==0) //пуст второй

P2=tmp0;

if (P1.Time1==0 && P2.Time1!=0) //пуст первый, не пуст второй

P1=tmp0;

if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P1.P)

{

putStackArray(&S1, P1); //поместили в стек то что было

P1=tmp0; //записали новый процесс

}

else if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P2.P)

{

putStackArray(&S2, P2); //поместили в стек то что было

P2=tmp0; //записали новый процесс

}

}

else

{

if (P1.Time1!=0) P1.Time1--;

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S1)) getStackArray(&S1, &P1);

else if (!isEmptyQueueArray(&F1)) getQueueArray(&F1, &tmp0);

}

if (P2.Time1!=0) //уменьшаем время в процессоре 1

P2.Time1--;

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S2)) getStackArray(&S2, &P2);

else if (!isEmptyQueueArray(&F1)) getQueueArray(&F1, &tmp0);

}

}

if (!isEmptyQueueArray(&F2))

{

if (P1.Time1==0 && P2.Time1 == 0) //процессоры пусты

P1=tmp0;

if (P1.Time1!=0 && P2.Time1==0) //пуст второй

P2=tmp0;

if (P1.Time1==0 && P2.Time1!=0) //пуст первый, не пуст второй

P1=tmp0;

if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P1.P)

{

// getQueueArray(&F0, &tmp0); //достанем из нее элемент

putStackArray(&S1, P1); //поместили в стек то что было

P1=tmp0; //записали новый процесс

}

else if (P1.Time1!=0 && P2.Time1!=0 && tmp0.P<P2.P)

{

putStackArray(&S2, P2); //поместили в стек то что было

P2=tmp0; //записали новый процесс

}

//getQueueArray(&F0, &tmp0);

}

else

{

if (P1.Time1!=0) P1.Time1--;

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S1)) getStackArray(&S1, &P1);

else if (!isEmptyQueueArray(&F2)) getQueueArray(&F2, &tmp0);

}

if (P2.Time1!=0) P2.Time1--;

else //иначе нужно вытащить что-то из стека

{

if(!isEmptyStackArray(&S2)) getStackArray(&S2, &P2);

else if (!isEmptyQueueArray(&F2)) getQueueArray(&F2, &tmp0);

}

}

if (i<n) { GenZapros(&tmp0); i++; }

FifoRaspr(tmp0, &F0, &F1, &F2);

}

Поступающие запросы ставятся в соответствующие приоритетам очереди. Сначала обрабатываются задачи из очереди F0. Задача из очереди F0 поступает в свободный процессор P1 или P2, если оба свободны, то в P1. Если очередь F0 пуста, то обрабатываются задачи из очереди F1. Задача из очереди F1 поступает в свободный процессор P1 или P2, если оба свободны, то в P1. Если очереди F0 и F1 пусты, то обрабатываются задачи из очереди F2. Задача из очереди F2 поступает в свободный процессор P1 или P2, если оба свободны, то в P1. Если процессоры заняты и поступает задача с более высоким приоритетом, чем обрабатываемая в одном из процессоров, то задача из процессора помещается в соответствующий стек, а поступающая — в процессор. Задача из стека помещается в соответствующий процессор, если он свободен, и очереди с задачами более высокого приоритета пусты.

В систему поступает 8 запросов:

|  |  |  |
| --- | --- | --- |
| Время | Объекты | Задачи |
| 0 | F0 | (Рома, 2) |
| F1 |  |
| F2 |  |
| S1 |  |
| S2 |  |
| P1 |  |
| P2 |  |
| 1 | F0 |  |
| F1 | (Костя, 3) |
| F2 |  |
| S1 |  |
| S2 |  |
| P1 | (Рома, 2) |
| P2 |  |
| 2 | F0 | (Вика, 2) |
| F1 |  |
| F2 |  |
| S1 |  |
| S2 |  |
| P1 | (Рома, 1) |
| P2 | (Костя, 3) |
| 3 | F0 | (Оля, 2) |
| F1 |  |
| F2 |  |
| S1 |  |
| S2 | (Костя, 3) |
| P1 | (Рома, 1) |
| P2 | (Вика, 2) |
| 4 | F0 | (Оля, 2) |
| F1 | (Антон, 3) |
| F2 |  |
| S1 |  |
| S2 | (Костя, 3) |
| P1 |  |
| P2 | (Вика, 3) |
| 5 | F0 |  |
| F1 | (Антон, 3) |
| F2 | (Никита, 2) |
| S1 |  |
| S2 | (Костя, 3) |
| P1 | (Оля, 2) |
| P2 | (Вика, 2) |
| 6 | F0 |  |
| F1 | (Антон, 3)(Рита, 2) |
| F2 | (Никита, 2) |
| S1 |  |
| S2 | (Костя, 3) |
| P1 | (Оля, 1) |
| P2 | (Вика, 1) |
| 7 | F0 | (Света, 2) |
| F1 | (Антон, 3)(Рита, 2) |
| F2 | (Никита, 2) |
| S1 |  |
| S2 | (Костя, 3) |
| P1 |  |
| P2 |  |
| 8 | F0 |  |
| F1 | (Антон, 3)(Рита, 2) |
| F2 | (Никита, 2) |
| S1 |  |
| S2 |  |
| P1 | (Света, 2) |
| P2 | (Костя, 3) |
| 9 | F0 |  |
| F1 | (Антон, 3)(Рита, 2) |
| F2 | (Никита, 2) |
| S1 |  |
| S2 |  |
| P1 | (Света, 1) |
| P2 | (Костя, 2) |
| 10 | F0 |  |
| F1 | (Антон, 3)(Рита, 2) |
| F2 | (Никита, 2) |
| S1 |  |
| S2 |  |
| P1 |  |
| P2 | (Костя, 1) |
| 11 | F0 |  |
| F1 | (Рита, 2) |
| F2 | (Никита, 2) |
| S1 |  |
| S2 |  |
| P1 | (Антон, 3) |
| P2 |  |
| 12 | F0 |  |
| F1 |  |
| F2 | (Никита, 2) |
| S1 |  |
| S2 |  |
| P1 | (Антон, 2) |
| P2 | (Рита, 2) |
| 13 | F0 |  |
| F1 |  |
| F2 | (Никита, 2) |
| S1 |  |
| S2 |  |
| P1 | (Антон, 1) |
| P2 | (Рита, 1) |
| 14 | F0 |  |
| F1 |  |
| F2 | (Никита, 2) |
| S1 |  |
| S2 |  |
| P1 |  |
| P2 |  |
| 15 | F0 |  |
| F1 |  |
| F2 |  |
| S1 |  |
| S2 |  |
| P1 | (Никита, 1) |
| P2 |  |
| 16 | F0 |  |
| F1 |  |
| F2 |  |
| S1 |  |
| S2 |  |
| P1 |  |
| P2 |  |