Белгородский Государственный Технологический Университет им. В. Г. Шухова

Кафедра программного обеспечения вычислительной техники  
и автоматизированных систем

## Лабораторная работа №6 по теме: «Структуры данных „стек“ и „очередь“»

Выполнил:  
студент группы ПВ-21  
Адаменко И. И.

Проверил:  
профессор  
Поляков В. К.

Белгород  
2013

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

## Задания

1. Для СД типа «стек» и «очередь» определить:
   1. Абстрактный уровень представления:
      1. Характер организованности и изменчивости.
      2. Набор допустимых операций.
   2. Физический уровень представления:
      1. Схему хранения.
      2. Объем памяти, занимаемый экземпляром СД.
      3. Формат внутреннего представления СД и его способ интерпретации.
      4. Характеристику допустимых значений.
      5. Тип доступа к элементам.
   3. Логический уровень представления СД. Способ описания СД и экземпляра СД на языке программирования.
2. Реализовать СД «стек» и «очередь» в соответствии с вариантом индивидуального задания в виде модуля на языке C.
3. Разработать программу на языке C, моделирующую вычислительную систему с постоянным шагом по времени (дискретное время) в соответствии с вариантом индивидуального задания, с использованием модулей, полученных в результате выполнения п. 2. Результат работы программы представить в виде таблицы. В первом столбце указывается время моделирования . Во втором — для каждого момента времени указываются имена объектов, а в третьем — задачи (имя, время), находящиеся в объектах.

### Задача

Система состоит из процессора , трех очередей и стека . В систему поступают запросы, каждый из которых можно представить записью:

|  |  |
| --- | --- |
| 1 | **typedef struct** { |
| 2 | **char** Name[10]; *// Имя запроса* |
| 3 | **unsigned** Time; *// Время обслуживания* |
| 4 | **char** P; *// Приоритет задачи: 0 — высший, 1 — средний, 2 — низший* |
| 5 | } TInquiry; |

Поступающие запросы ставятся в соответствующие приоритетам очереди. Сначала обрабатываются задачи из очереди . Если она пуста, можно обрабатывать задачи из очереди . Если и она пуста, то можно обрабатывать задачи из очереди . Если все очереди пусты, то система находится в ожидании поступающих задач (процессор свободен), либо в режиме обработки предыдущей задачи (процессор занят). Если поступает задача с более высоким приоритетом, чем обрабатываемая в данный момент, то обрабатываемая помещается в стек и может обрабатываться тогда и только тогда, когда все задачи с более высоким приоритетом уже обработаны.

### Модуль для реализации стека

Стек на массиве в динамической памяти.

|  |  |
| --- | --- |
| 01 | **#ifndef** STACK\_ARRAY\_H |
| 02 | **#define** STACK\_ARRAY\_H |
| 03 |  |
| 04 | **const short** StackSize = 1000; |
| 05 | **const short** StackOk = 0; |
| 06 | **const short** StackEmpty = 1; |
| 07 | **const short** StackFull = 2; |
| 08 | **static short** StackError; *// Переменная ошибок* |
| 09 |  |
| 10 | **typedef** ... BaseType; *// Определить тип элемента* |
| 11 |  |
| 12 | **typedef** struct { |
| 13 | BaseType Buf[StackSize]; |
| 14 | **unsigned** Top; *// Указывает элемент — на вершину стека* |
| 15 | } Stack; |
| 16 |  |
| 17 | *// Инициализация стека* |
| 18 | **void** InitStack(Stack \*S); |
| 19 |  |
| 20 | *// Вставка в стек* |
| 21 | **void** PutStack(Stack \*S, BaseType E); |
| 22 |  |
| 23 | *// Взятие из стека* |
| 24 | **void** GetStack(Stack \*S, BaseType \*E); |
| 25 |  |
| 26 | *// Чтение из стека* |
| 27 | **void** ReadStack(Stack \*S, BaseType \*E); |
| 28 |  |
| 29 | *// Проверка стека на пустоту* |
| 30 | **int** EmptyStack(Stack \*S); |
| 31 |  |
| 32 | **#endif** |

### Модуль для реализации очереди

Очередь на ОЛС. «Хвост» очереди — последний, а «голова» — первый элемент ОЛС.

|  |  |
| --- | --- |
| 01 | **#ifndef** QUEUE\_LIST\_H |
| 02 | **#define** QUEUE\_LIST\_H |
| 03 |  |
| 04 | **#include** *"SLL.h"* |
| 05 |  |
| 06 | **const short** QueueOk = ListOk; |
| 07 | **const short** QueueEmpty = ListEmpty; |
| 08 | **const short** QueueNotMem = ListNotMem; |
| 09 | **static short** QueueError; *// Переменная ошибок* |
| 10 |  |
| 11 | **typedef** List Queue; |
| 12 |  |
| 13 | *// Инициализация очереди* |
| 14 | **void** InitQueue(Queue \*Q); |
| 15 |  |
| 16 | *// Вставка в очередь* |
| 17 | **void** PutQueue(Queue \*Q, BaseType E); |
| 18 |  |
| 19 | *// Взятие из очереди* |
| 20 | **void** GetQueue(Queue \*Q, BaseType \*E); |
| 21 |  |
| 22 | *// Чтение из очереди* |
| 23 | **void** ReadQueue(Queue \*Q, BaseType \*E); |
| 24 |  |
| 25 | *// Очистка очереди* |
| 26 | **void** DoneQueue(Queue \*Q); |
| 27 |  |
| 28 | *// Проверка на пустоту* |
| 29 | **int** EmptyQueue(Queue \*Q); |
| 30 |  |
| 31 | **#endif** |

## Характеристика СД типа «стек»

1. Абстрактный уровень представления СД:
   1. По характеру организованности и изменчивости данный тип упорядоченный и динамический.
   2. К типу применимы такие операции, как: инициализация, включение элемента, исключение элемента, чтение текущего элемента и пр.
2. Физический уровень представления СД:
   1. Структура данных хранится последовательно.
   2. Размер памяти, занимаемый каждым экземпляром СД зависит от максимального возможного количества элементов в нее входящих (и от выбранной схемы хранения отдельного элемента).
   3. В зависимости от схемы хранения, меняется и формат внутреннего представления СД.
   4. Допустимыми являются все значения типа данных, определенного как базовый.
   5. Тип доступа к элементам данной СД последовательный.
3. На логическом уровне представления СД, то есть, на языке программирования С экземпляр СД описывается следующим образом: Stack L;

## Характеристика СД типа «очередь»

1. Абстрактный уровень представления СД:
   1. По характеру организованности и изменчивости данный тип упорядоченный и динамический.
   2. К типу применимы такие операции, как: инициализация, включение элемента, исключение элемента, чтение текущего элемента и пр.
2. Физический уровень представления СД:
   1. Структура данных хранится связно.
   2. Размер памяти, занимаемый каждым экземпляром СД зависит от максимального возможного количества элементов в нее входящих (и от выбранной схемы хранения отдельного элемента).
   3. В зависимости от схемы хранения, меняется и формат внутреннего представления СД.
   4. Допустимыми являются все значения типа данных, определенного как базовый.
   5. Тип доступа к элементам данной СД последовательный.
3. На логическом уровне представления СД, то есть, на языке программирования С экземпляр СД описывается следующим образом: Queue L;

## Текст модуля для реализации СД типа «стек»

### StackArray.h

|  |  |
| --- | --- |
| 01 | **#ifndef** STACK\_ARRAY\_H |
| 02 | **#define** STACK\_ARRAY\_H |
| 03 |  |
| 04 | **#include** <cstdio> |
| 05 | **#include** <cstdlib> |
| 06 | **#include** *"TInquiry.h"* |
| 07 |  |
| 08 | **const** **short** StackSize = 1000; |
| 09 | **const** **short** StackOk = 0; |
| 10 | **const** **short** StackEmpty = 1; |
| 11 | **const** **short** StackFull = 2; |
| 12 | **static** **short** StackError; |
| 13 |  |
| 14 | **typedef** TInquiry BaseType; |
| 15 |  |
| 16 | **typedef** **struct** { |
| 17 | BaseType Buf[StackSize]; |
| 18 | unsigned Top; |
| 19 | } Stack; |
| 20 |  |
| 21 | *// Инициализация стека* |
| 22 | **void** InitStack(Stack \*S); |
| 23 |  |
| 24 | *// Вставка в стек* |
| 25 | **void** PutStack(Stack \*S, BaseType E); |
| 26 |  |
| 27 | *// Взятие из стека* |
| 28 | **void** GetStack(Stack \*S, BaseType \*E); |
| 29 |  |
| 30 | *// Чтение из стека* |
| 31 | **void** ReadStack(Stack \*S, BaseType \*E); |
| 32 |  |
| 33 | *// Вывод содержимого стека* |
| 34 | **void** PrintStack(Stack \*S); |
| 35 |  |
| 36 | *// Проверка стека на заполненность* |
| 37 | **bool** FullStack(Stack \*S); |
| 38 |  |
| 39 | *// Проверка стека на пустоту* |
| 40 | **bool** EmptyStack(Stack \*S); |
| 41 |  |
| 42 | **#endif** |

### StackArray.cpp

|  |  |
| --- | --- |
| 01 | **#include** *"StackArray.h"* |
| 02 |  |
| 03 | *// Инициализация стека* |
| 04 | **void** InitStack(Stack \*S) { |
| 05 | S->Top = StackSize; |
| 06 | StackError = StackOk; |
| 07 | }; |
| 08 |  |
| 09 | *// Вставка в стек* |
| 10 | **void** PutStack(Stack \*S, BaseType E) { |
| 11 | **if** (S->Top == 0) { |
| 12 | StackError = StackFull; |
| 13 | **return**; |
| 14 | } |
| 15 |  |
| 16 | S->Buf[--S->Top] = E; |
| 17 | }; |
| 18 |  |
| 19 | *// Взятие из стека* |
| 20 | **void** GetStack(Stack \*S, BaseType \*E) { |
| 21 | **if** (S->Top == StackSize) { |
| 22 | StackError = StackEmpty; |
| 23 | **return**; |
| 24 | } |
| 25 |  |
| 26 | \*E = S->Buf[S->Top++]; |
| 27 | }; |
| 28 |  |
| 29 | *// Чтение из стека* |
| 30 | **void** ReadStack(Stack \*S, BaseType \*E) { |
| 31 | **if** (S->Top == StackSize) { |
| 32 | StackError = StackEmpty; |
| 33 | **return**; |
| 34 | } |
| 35 |  |
| 36 | \*E = S->Buf[S->Top]; |
| 37 | } |
| 38 |  |
| 39 | *// Проверка стека на заполненность* |
| 40 | **bool** FullStack(Stack \*S) { |
| 41 | **if** (S->Top == 0) { |
| 42 | StackError = StackFull; |
| 43 | **return** true; |
| 44 | } |
| 45 | **return** **false**; |
| 46 | } |
| 47 |  |
| 48 | *// Вывод содержимого стека* |
| 49 | **void** PrintStack(Stack \*S) { |
| 50 | BaseType e; |
| 51 | Stack tmpStack; |
| 52 |  |
| 53 | InitStack(&tmpStack); |
| 54 |  |
| 55 | **while** (!EmptyStack(S)) { |
| 56 | GetStack(S, &e); |
| 57 | PutStack(&tmpStack, e); |
| 58 | } |
| 59 |  |
| 60 | **while** (!EmptyStack(&tmpStack)) { |
| 61 | GetStack(&tmpStack, &e); |
| 62 | **printf**(*"\t--(%s, %d, %d)\n"*, e.Name, e.Time, e.P); |
| 63 | PutStack(S, e); |
| 64 | } |
| 65 | } |
| 66 |  |
| 67 | *// Проверка стека на пустоту* |
| 68 | **bool** EmptyStack(Stack \*S) { |
| 69 | **if** (S->Top == StackSize) { |
| 70 | StackError = StackEmpty; |
| 71 | **return** **true**; |
| 72 | } |
| 73 | **return** **false**; |
| 74 | }; |

## Текст модуля для реализации СД типа «очередь»

### QueueList.h

|  |  |
| --- | --- |
| 01 | **#ifndef** QUEUE\_LIST\_H |
| 02 | **#define** QUEUE\_LIST\_H |
| 03 |  |
| 04 | **#include** *"SLL.h"* |
| 05 |  |
| 06 | **const** **short** QueueOk = ListOk; |
| 07 | **const** **short** QueueEmpty = ListEmpty; |
| 08 | **const** **short** QueueNotMem = ListNotMem; |
| 09 | **static** **short** QueueError; |
| 10 |  |
| 11 | **typedef** List Queue; |
| 12 |  |
| 13 | *// Инициализация очереди* |
| 14 | **void** InitQueue(Queue \*Q); |
| 15 |  |
| 16 | *// Вставка в очередь* |
| 17 | **void** PutQueue(Queue \*Q, BaseType E); |
| 18 |  |
| 19 | *// Взятие из очереди* |
| 20 | **void** GetQueue(Queue \*Q, BaseType \*E); |
| 21 |  |
| 22 | *// Чтение из очереди* |
| 23 | **void** ReadQueue(Queue \*Q, BaseType \*E); |
| 24 |  |
| 25 | *// Вывод содержимого очереди* |
| 26 | **void** PrintQueue(Queue \*Q); |
| 27 |  |
| 28 | *// Очистка очереди* |
| 29 | **void** DoneQueue(Queue \*Q); |
| 30 |  |
| 31 | *// Проверка на пустоту* |
| 32 | **bool** EmptyQueue(Queue \*Q); |
| 33 |  |
| 34 | **#endif** |

### QueueList.cpp

|  |  |
| --- | --- |
| 01 | **#include** *"QueueList.h"* |
| 02 |  |
| 03 | *// Инициализация очереди* |
| 04 | **void** InitQueue(Queue \*Q) { |
| 05 | InitList(Q); |
| 06 | QueueError = GetListError(); |
| 07 | } |
| 08 |  |
| 09 | *// Вставка в очередь* |
| 10 | **void** PutQueue(Queue \*Q, BaseType E) { |
| 11 | BeginPtr(Q); |
| 12 | PutList(Q, E); |
| 13 | QueueError = GetListError(); |
| 14 | } |
| 15 |  |
| 16 | *// Взятие из очереди* |
| 17 | **void** GetQueue(Queue \*Q, BaseType \*E) { |
| 18 | EndPtr(Q); |
| 19 | GetList(Q, E); |
| 20 | QueueError = GetListError(); |
| 21 | } |
| 22 |  |
| 23 | *// Чтение из очереди* |
| 24 | **void** ReadQueue(Queue \*Q, BaseType \*E) { |
| 25 | ReadList(Q, E); |
| 26 | QueueError = GetListError(); |
| 27 | } |
| 28 |  |
| 29 | *// Вывод содержимого очереди* |
| 30 | **void** PrintQueue(Queue \*S) { |
| 31 | BaseType e; |
| 32 | Queue tmpQueue; |
| 33 |  |
| 34 | InitQueue(&tmpQueue); |
| 35 |  |
| 36 | **while** (!EmptyQueue(S)) { |
| 37 | GetQueue(S, &e); |
| 38 | PutQueue(&tmpQueue, e); |
| 39 | } |
| 40 |  |
| 41 | **while** (!EmptyQueue(&tmpQueue)) { |
| 42 | GetQueue(&tmpQueue, &e); |
| 43 | **printf**(*"\t--(%s, %d, %d)\n",* e.Name, e.Time, e.P); |
| 44 | PutQueue(S, e); |
| 45 | } |
| 46 | } |
| 47 |  |
| 48 | *// Очистка очереди* |
| 49 | **void** DoneQueue(Queue \*Q) { |
| 50 | ClearList(Q); |
| 51 | QueueError = GetListError(); |
| 52 | } |
| 53 |  |
| 54 | *// Проверка на пустоту* |
| 55 | **bool** EmptyQueue(Queue \*Q) { |
| 56 | **return** EmptyList(Q); |
| 57 | } |

## Текст программы для решения задачи

### TInquiry.h

|  |  |
| --- | --- |
| 01 | **#ifndef** TINQUIRY |
| 02 | **#define** TINQUIRY |
| 03 |  |
| 04 | **typedef** **struct** { |
| 05 | **char** Name[10]; |
| 06 | **unsigned** Time; |
| 07 | **char** P; |
| 08 | } TInquiry; |
| 09 |  |
| 10 | **#endif** |

### main.h

|  |  |
| --- | --- |
| 01 | **#include** <cstdio> |
| 02 | **#include** <cstdlib> |
| 03 | **#include** <ctime> |
| 04 | **#include** <random> |
| 05 |  |
| 06 | **#include** *"StackArray.h"* |
| 07 | **#include** *"QueueList.h"* |
| 08 | **#include** *"TInquiry.h"* |
| 09 |  |
| 10 | **void** generateName(**char**\* string); |
| 11 | TInquiry generator(); |
| 12 | **void** dump(Queue \*F0, Queue \*F1, Queue \*F2, Stack \*S, TInquiry currentTask,  **int** number); |

### main.cpp

|  |  |
| --- | --- |
| 001 | **#include** *"main.h"* |
| 002 |  |
| 003 | **int** main(**int** argc, **char** **const** \*argv[]) { |
| 004 | Queue F0, F1, F2; |
| 005 | Stack S; |
| 006 | TInquiry currentTask, tmpTask; |
| 007 | **int** isFree = 1; |
| 008 |  |
| 009 | InitQueue(&F0); |
| 010 | InitQueue(&F1); |
| 011 | InitQueue(&F2); |
| 012 | InitStack(&S); |
| 013 |  |
| 014 | **for** (**int** i = 0; i < 15; i++) { |
| 015 | tmpTask = generator(); |
| 016 |  |
| 017 | **switch** (tmpTask.P) { |
| 018 | **case** 0: PutQueue(&F0, tmpTask); |
| 019 | **break**; |
| 020 | **case** 1: PutQueue(&F1, tmpTask); |
| 021 | **break**; |
| 022 | **case** 2: PutQueue(&F2, tmpTask); |
| 023 | **break**; |
| 024 | } |
| 025 |  |
| 026 | **if** (isFree == 0) { |
| 027 | **if** (currentTask.P > tmpTask.P) { |
| 028 | PutStack(&S, currentTask); |
| 029 |  |
| 030 | **switch** (tmpTask.P) { |
| 031 | **case** 0: GetQueue(&F0, &currentTask); |
| 032 | **break**; |
| 033 | **case** 1: GetQueue(&F1, &currentTask); |
| 034 | **break**; |
| 035 | **case** 2: GetQueue(&F2, &currentTask); |
| 036 | **break**; |
| 037 | } |
| 038 | } |
| 039 | } |
| 040 | **else** { |
| 041 | **if** (!EmptyQueue(&F0)) { |
| 042 | GetQueue(&F0, &currentTask); |
| 043 | } |
| 044 | **else** { |
| 045 | **if** (!EmptyStack(&S)) { |
| 046 | ReadStack(&S, &tmpTask); |
| 047 |  |
| 048 | **if** (tmpTask.P == 1) { |
| 049 | GetStack(&S, &currentTask); |
| 050 | } |
| 051 | **else** **if** (!EmptyQueue(&F1)) { |
| 052 | GetQueue(&F1, &currentTask); |
| 053 | } |
| 054 | **else** { |
| 055 | GetStack(&S, &currentTask); |
| 056 | } |
| 057 | } |
| 058 | **else** **if** (!EmptyQueue(&F1)) { |
| 059 | GetQueue(&F1, &currentTask); |
| 060 | } |
| 061 | **else** **if** (!EmptyQueue(&F2)) { |
| 062 | GetQueue(&F2, &currentTask); |
| 063 | } |
| 064 | } |
| 065 | } |
| 066 |  |
| 067 | dump(&F0, &F1, &F2, &S, currentTask, i); |
| 068 | **if** (--currentTask.Time <= 0) { |
| 069 | isFree = 1; |
| 070 | } |
| 071 | **else** { |
| 072 | isFree = 0; |
| 073 | } |
| 074 | } |
| 075 |  |
| 076 | getchar(); |
| 077 | **return** 0; |
| 078 | } |
| 079 |  |
| 080 | **void** dump(Queue \*F0, Queue \*F1, Queue \*F2, Stack \*S, TInquiry currentTask, **int** number) { |
| 081 | **printf**(*"%d"*, number); |
| 082 |  |
| 083 | **switch** (number) { |
| 084 | **case** 0: **printf**(*" "*); |
| 085 | **break**; |
| 086 | **case** 1: **printf**(*"%s ",* *"st"*); |
| 087 | **break**; |
| 088 | **case** 2: **printf**(*"%s ",* *"nd"*); |
| 089 | **break**; |
| 090 | **case** 3: **printf**(*"%s ", "rd"*); |
| 091 | **break**; |
| 092 | **default**: **printf**(*"%s ", "th"*); |
| 093 | **break**; |
| 094 | } |
| 095 |  |
| 096 | **printf**(*"%s\n", "time:"*); |
| 097 |  |
| 098 | **printf**(*"\t%s\n", "F0:"*); |
| 099 | PrintQueue(F0); |
| 100 |  |
| 101 | **printf**(*"\t%s\n", "F1:"*); |
| 102 | PrintQueue(F1); |
| 103 |  |
| 104 | **printf**(*"\t%s\n", "F2:"*); |
| 105 | PrintQueue(F2); |
| 106 |  |
| 107 | **printf**(*"\t%s\n", "S:"*); |
| 108 | PrintStack(S); |
| 109 |  |
| 110 | **printf**(*"\t%s\n", "P:"*); |
| 111 | **if** (currentTask.Time > 0) { |
| 112 | **printf**(*"\t--(%s, %d, %d)\n",* currentTask.Name,   currentTask.Time, currentTask.P); |
| 113 | } |
| 114 | } |
| 115 |  |
| 116 | TInquiry generator() { |
| 117 | TInquiry task; |
| 118 | std::random\_device rd; |
| 119 |  |
| 120 | generateName(task.Name); |
| 121 | task.Time = rd() % 5 + 1; |
| 122 | task.P = rd() % 3; |
| 123 |  |
| 124 | **return** task; |
| 125 | } |
| 126 |  |
| 127 | **void** generateName(**char** \*name) { |
| 128 | **int** range = *'Z'* - *'A';* |
| 129 | std::random\_device rd; |
| 130 |  |
| 131 | **for** (**int** i = 0; i < 9; i++) { |
| 132 | name[i] = *'A'* + rd() % range; |
| 133 | } |
| 134 |  |
| 135 | name[9] = *'\0';* |
| 136 | } |

## Результаты работы программы

*Пример задачи: (Имя, Оставшееся время выполнения, Приоритет)*

|  |  |  |
| --- | --- | --- |
| Время | Объекты | Задачи |
| 0 |  |  |
|  |  |
|  |  |
|  |  |
|  | (IKNWWUOSP, 4, 2) |
| 1 |  |  |
|  |  |
|  | (TAIEHVFTD, 4, 2) |
|  |  |
|  | (IKNWWUOSP, 3, 2) |
| 2 |  |  |
|  |  |
|  | (TAIEHVFTD, 4, 2) |
|  | (IKNWWUOSP, 2, 2) |
|  | (ANHCHPUDH, 5, 1) |
| 3 |  |  |
|  |  |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2) |
|  | (IKNWWUOSP, 2, 2) |
|  | (ANHCHPUDH, 4, 1) |
| 4 |  |  |
|  |  |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (NAWXUHBIS, 5, 0) |
| 5 |  | (FUMKLVAGQ, 2, 0) |
|  |  |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (NAWXUHBIS, 4, 0) |
| 6 |  | (FUMKLVAGQ, 2, 0), (KWAUCOJHY, 2, 0) |
|  |  |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (NAWXUHBIS, 3, 0) |
| 7 |  | (FUMKLVAGQ, 2, 0), (KWAUCOJHY, 2, 0) |
|  | (IDSJWBJYV, 1, 1) |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (NAWXUHBIS, 2, 0) |
| 8 |  | (FUMKLVAGQ, 2, 0), (KWAUCOJHY, 2, 0), (OMERINXUY, 1, 0) |
|  | (IDSJWBJYV, 1, 1) |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (NAWXUHBIS, 1, 0) |
| 9 |  | (KWAUCOJHY, 2, 0), (OMERINXUY, 1, 0) |
|  | (IDSJWBJYV, 1, 1) |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2), (WWUQDDLIO, 4, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (FUMKLVAGQ, 2, 0) |
| 10 |  | (KWAUCOJHY, 2, 0), (OMERINXUY, 1, 0) |
|  | (IDSJWBJYV, 1, 1), (TJDDUKNEP, 3, 1) |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2), (WWUQDDLIO, 4, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (FUMKLVAGQ, 1, 0) |
| 11 |  | (OMERINXUY, 1, 0) |
|  | (IDSJWBJYV, 1, 1), (TJDDUKNEP, 3, 1) |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2), (WWUQDDLIO, 4, 2), (XAKKAJIAK, 2, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (KWAUCOJHY, 2, 0) |
| 12 |  | (OMERINXUY, 1, 0) |
|  | (IDSJWBJYV, 1, 1), (TJDDUKNEP, 3, 1) |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2), (WWUQDDLIO, 4, 2), (XAKKAJIAK, 2, 2), (WGTDAFWKA, 4, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (KWAUCOJHY, 1, 0) |
| 13 |  |  |
|  | (IDSJWBJYV, 1, 1), (TJDDUKNEP, 3, 1), (ERLEHYOMM, 2, 1) |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2), (WWUQDDLIO, 4, 2), (XAKKAJIAK, 2, 2), (WGTDAFWKA, 4, 2) |
|  | (IKNWWUOSP, 2, 2), (ANHCHPUDH, 3, 1) |
|  | (OMERINXUY, 1, 0) |
| 14 |  |  |
|  | (IDSJWBJYV, 1, 1), (TJDDUKNEP, 3, 1), (ERLEHYOMM, 2, 1) |
|  | (TAIEHVFTD, 4, 2), (NROUTYGRR, 5, 2), (WWUQDDLIO, 4, 2), (XAKKAJIAK, 2, 2), (WGTDAFWKA, 4, 2), (BQNAKMUWO, 5, 2) |
|  | (IKNWWUOSP, 2, 2) |
|  | (ANHCHPUDH, 3, 1) |