Лекция 15

Динамические структуры данных: очереди и стеки

Очереди

Очередь – динамическая структура данных с упорядоченным доступом к элементам, функционирующая по принципу FIFO.

First

In

First

Out

Последовательное хранение

Типы и переменные

```
typedef struct{
  TYPE *list;
  int size,count,head,tail;
} QUEUE;
```

Создание очереди

```
int Create(QUEUE *queue,int sz)
{
  queue->list = (TYPE*)calloc(sz,sizeof(TYPE));
  if(!queue->list) return 0;
  queue->size = sz;
  queue->count = queue->head = queue->tail = 0;
  return 1;
}
```

Удаление очереди

```
void Clear(QUEUE *queue)
{
  if(queue->list) free(queue->list);
  queue->list = NULL;
  queue->size = queue->count = 0;
  queue->head = queue->tail = 0;
}
```

Помещение элемента в очередь

```
int Put(QUEUE *queue, TYPE val)
{
  if(queue->count == queue->size) return 0;
  queue->list[queue->tail++] = val;
  if(queue->tail == queue->size) queue->tail = 0;
  queue->count++;
  return 1;
}
```

Изъятие элемента из очереди

```
int Get(QUEUE *queue, TYPE *val)
{
  if(queue->count == 0) return 0;
  *val = queue->list[queue->head++];
  if(queue->head == queue->size) queue->head = 0;
  queue->count--;
  return 1;
}
```

Пример

Реализовать программу, которая в интерактивном режиме запрашивает у пользователя команду и выполняет ее.

Команды:

- exit – завершение программы,
- put N – помещение N в очередь,
- get – изъять элемент из очереди и вывести его значение на экран.

Дополнительно программа должна выводить сообщения о невозможности выполнения операции

Программа

```
int main(int argc, char *argv[])
{
  QUEUE q;
  Create(&q,20);
  while(1){
    char cmd[21]; int value;
    printf(">: "); gets(cmd);
    if(strncmp(cmd,"exit",4)==0) break;
    if(strncmp(cmd,"put",3)==0){
      char *tail = NULL;
      value = strtol(&cmd[3],&tail,10);
      if(strlen(tail)==0){
          if(!Put(&q,value)) puts("Очередь заполнена!");
      } else puts("Некорректная команда");
    }else if(strcmp(cmd,"get")==0){
      if(Get(&q,&value)) printf("%d\n",value);
        else puts("Очередь пуста!");
    } else puts("Неизвестная команда!");
  }
  Clear(&q);
  return 0;
}
```

Связанное хранение

Типы и переменные:

```
typedef struct _ELEMENT{
  TYPE value;
  struct _ELEMENT *next;
} ELEMENT;
typedef struct{
  ELEMENT *head, *tail;
}QUEUE;
```

Создание и удаление очереди

```
void Create(QUEUE *queue)
{
  queue->head = queue->tail = NULL;
}

void Clear(QUEUE *queue)
{
  while(queue->head){
    ELEMENT *tmp = queue->head;
    queue->head = tmp->next;
    free(tmp);
  }
  queue->tail = NULL;
}
```

Помещение элемента в очередь

```
int Put(QUEUE *queue, TYPE val)
{
  ELEMENT *tmp = (ELEMENT*)malloc(sizeof(ELEMENT));
  if(!tmp) return 0;
  tmp->next = NULL;
  tmp->value = val;
  if(queue->tail) queue->tail->next = tmp;
  queue->tail = tmp;
  if(!queue->head) queue->head = queue->tail;
  return 1;
}
```

Изъятие элемента из очереди

```
int Get(QUEUE *queue, TYPE *val)
{
  if(!queue->head) return 0;
  ELEMENT *tmp = queue->head;
  queue->head = tmp->next;
  *val = tmp->value;
  free(tmp);
  if(!queue->head) queue->tail = NULL;
  return 1;
}
```

Стек

Стек – динамическая структура данных с упорядоченным доступом к элементам, функционирующая по принципу LIFO.

Last

In

First

Out

Последовательное хранение

Типы и переменные:

```
typedef struct{
  TYPE *list;
  int size,head;
} STACK;
```

Создание стека

```
int Create(STACK *stack, int sz)
{
  stack->list = (TYPE*)calloc(sz,sizeof(TYPE));
  if(!stack->list) return 0;
  stack->head = -1;
  stack->size = sz;
  return 1;
}
```

Удаление стека

```
void Clear(STACK *stack)
{
  if(stack->list) free(stack->list);
  stack->list = NULL;
  stack->head = 0;
  stack->size = 0;
}
```

Помещение элемента в стек

```
int Push(STACK *stack, TYPE val)
{
  if(stack->head == stack->size-1) return 0;
  stack->list[++stack->head] = val;
  return 1;
}
```

Изъятие элемента из стека

```
int Pop(STACK *stack, TYPE *val)
{
  if(stack->head == -1) return 0;
  *val = stack->list[stack->head--];
  return 1;
}
```

Пример

Реализовать программу, которая в интерактивном режиме запрашивает у пользователя команду и выполняет ее.

Команды:

- exit – завершение программы,
- push N – помещение N в стек,
- pop – изъять элемент из стека и вывести его значение на экран.

Дополнительно программа должна выводить сообщения о невозможности выполнения операции

Программа

```
int main(int argc, char *argv[])
{
  STACK q;
  Create(&q,20);
  while(1){
    char cmd[21]; int value;
    printf(">: "); gets(cmd);
    if(strncmp(cmd,"exit",4)==0) break;
    if(strncmp(cmd,"push",4)==0){
      char *tail = NULL;
      value = strtol(&cmd[4],&tail,10);
      if(strlen(tail)==0){
          if(!Push(&q,value)) puts("Стек заполнен!");
      } else puts("Некорректная команда");
    }else if(strcmp(cmd,"get")==0){
      if(Pop(&q,&value)) printf("%d\n",value);
        else puts("Стек пуст!");
    } else puts("Неизвестная команда!");
  }
  Clear(&q);
  return 0;
}
```

Связанное хранение

Типы и переменные:

```
typedef struct _ELEMENT{
  TYPE value;
  struct _ELEMENT *next;
} ELEMENT;
typedef ELEMENT* STACK;
```

Создание и удаление стека

```
void Create(STACK *stack)
{
  *stack = NULL;
}

void Clear(STACK *stack)
{
  while(*stack){
    ELEMENT *tmp = *stack;
    *stack = tmp->next;
    free(tmp);
  }
}
```

Помещение элемента в стек

```
int Push(STACK *stack, TYPE val)
{
  ELEMENT *tmp = (ELEMENT*)malloc(sizeof(ELEMENT));
  if(!tmp) return 0;
  tmp->next = *stack;
  tmp->value = val;
  *stack = tmp;
  return 1;
}
```

Изъятие элемента из стека

```
int Pop(STACK *stack, TYPE *val)
{
  if(!*stack) return 0;
  ELEMENT *tmp = *stack;
  *stack = tmp->next;
  *val = tmp->value;
  free(tmp);
  return 1;
}
```