Абстракции

u.to/72D_Gg

Лекция 4, 26 февраля, 2021

Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

dseverov@mail.mipt.ru

Обратная связь: **u.to/7Wn7Gg**

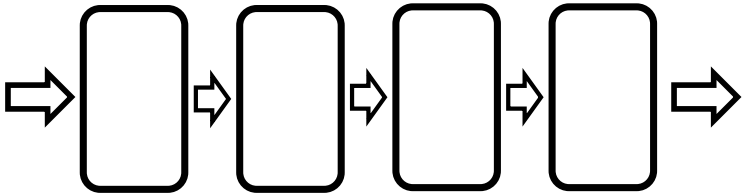
Введение

- Почему типы «абстрактные»?
- Реализации массивами и структурами

СПИСОК, ОЧЕРЕДЬ, СТЕК

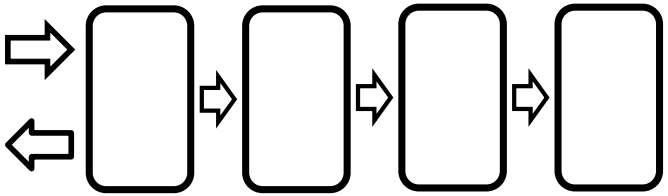
Очередь

FIFO – First In First Out



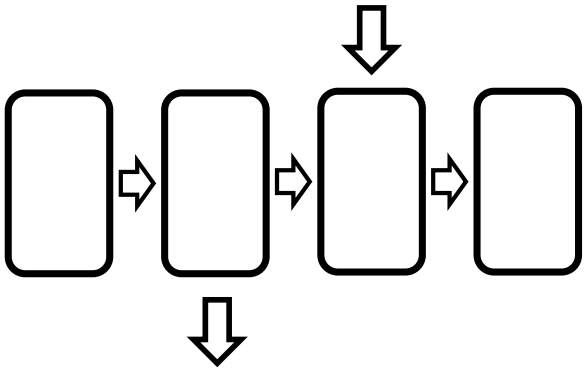
Стек

LIFO – Last In First Out



Список

Casual In Casual Out



Реализуем

- Что ?
  - Компоненты
  - Конструкция
  - Состояние
  - Действия
- Чем ?
  - Массивами
  - Структурами
  - Абстракциями

РЕАЛИЗАЦИЯ МАССИВАМИ

Стек. Компоненты

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

typedef int T;
typedef struct Stack *Link_Stack;

inline void ERR(const char* s) {
          fputs(s,stderr);  exit(1); }
```

Стек. Конструкция

```
struct Stack { int NumEl, Size; T Items[]; };

Link_Stack Stack_ini(int N) {
  Link_Stack a = (Link_Stack)malloc(
      sizeof(struct Stack) + N*sizeof(T));
  a->NumEl=0; a->Size=N; return a; }

Link_Stack Stack_free(Link_Stack a) {
  free(a); return NULL; }
```

Стек. Состояние

```
int Stack_Is_Empty(Link_Stack a) {
 return a->NumEl==0; }

int Stack_ Is_Full(Link_Stack a) {
 return a->NumEl==a->Size; }
```

Стек. Действия

```
void Stack_push(Link_Stack a, T New_el) {
  if(Stack_Is_Full(a))
      ERR("Stack::push: stack is full");
  a->Items[a->NumEl++]=New_el; }

T Stack_pop(Link_Stack a) {
  if(Stack_Is_Empty(a))
      ERR("Stack::pop: stack is empty");
  return a->Items [--a->NumEl]; }
```

```c
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
inline void ERR(const char* s) { fputs(s,stderr);  exit(1); }

typedef int T;
typedef struct Stack *Link_Stack;
struct Stack { int NumEl, Size; T Items[]; };

Link_Stack Stack_ini(int N) { Link_Stack a = (Link_Stack)malloc(sizeof(struct Stack) + N*sizeof(T));
                              a->NumEl=0; a->Size=N; return a; }
Link_Stack Stack_free(Link_Stack a) { free(a); return NULL; }
int Stack_Is_Empty(const Link_Stack a) { return a->NumEl==0; }
int Stack_Is_Full(const Link_Stack a) { return a->NumEl==a->Size; }
void Stack_push(Link_Stack a, T New_el) { if(Stack_Is_Full(a)) ERR("Stack::push: stack is full");
                              a->Items[a->NumEl++]=New_el; }
T Stack_pop(Link_Stack a) { if(Stack_Is_Empty(a)) ERR("Stack::pop: stack is empty"); return a->Items [--a->NumEl]; }

int main() {
int k,N;
    scanf("%d",&N);
Link_Stack a = Stack_ini(N);
    while(scanf("%d",&k)!=EOF) Stack_push(a,k);
    while(!Stack_Is_Empty(a)) { k=Stack_pop(a); printf("%d ",k); }
    a=Stack_free(a);

    return 0;
}
```

```c
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
inline void ERR(const char* s) { fputs(s,stderr);  exit(1); }

typedef int T;
typedef struct Stack *Link_Stack;
struct Stack { int NumEl, Size; T Items[

Link_Stack Stack_ini(int N) { Link_Stack
                              a->NumEl=0
Link_Stack Stack_free(Link_Stack a) { free(a); return NULL; }
int Stack_Is_Empty(const Link_Stack a) { return a->NumEl==0; }
int Stack_Is_Full(const Link_Stack a) { return a->NumEl==a->Size; }
void Stack_push(Link_Stack a, T New_el) { if(Stack_Is_Full(a)) ERR("Stack::push: stack is full");
                              a->Items[a->NumEl++]=New_el; }
T Stack_pop(Link_Stack a) { if(Stack_Is_Empty(a)) ERR("Stack::pop: stack is empty"); return a->Items [--a->NumEl]; }

int main() {
int k,N;
    scanf("%d",&N);
Link_Stack a = Stack_ini(N);
    while(scanf("%d",&k)!=EOF) Stack_push(a,
    while(!Stack_Is_Empty(a)) { k=Stack_pop(
    a=Stack_free(a);

    return 0;
}
```

```
5
1 2 3 4 5 6
Stack::push: stack is full
```

```
10
1 2 3 4 5 6 7 8 9 0
^Z
0 9 8 7 6 5 4 3 2 1
```

Очередь. Компоненты

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>

typedef int T;

typedef struct Queue *Link_Queue;

inline void ERR(const char* s) {
  fputs(s,stderr);  exit(1); }
```

Очередь. Конструкция

```
struct Queue { int NumEl, Size, P; T Items[]; };

Link_Queue Queue_ini(int N) { Link_Queue a =
  (Link_Queue)malloc(
      sizeof(struct Queue) + N*sizeof(T));
  a->Size=N; a->NumEl=a->P=0; return a; }

Link_Queue Queue_free(Link_Queue a) { free(a);
  return NULL; }
```

Очередь. Состояние

```
int Queue_Is_Empty(Link_Queue a) {
  return a->NumEl==0; }

int Queue_Is_Full(Link_Queue a) {
  return a->NumEl==a->Size; }
```

Очередь. Действия

```
void Queue_put(Link_Queue a, T New_el) {
  if(Queue_Is_Full(a))
      ERR("Queue::put: queue is full");
    a->Items[a->P++]=New_el; a->NumEl++;
  if(a->P==a->Size) a->P=0; }

T Queue_get(Link_Queue a) {
  if(Queue_Is_Empty(a))
      ERR("Queue::get: queue is empty");
    return a->Items[a->P - a->NumEl-- +
   (a->P <= a->NumEl?a->Size:0)]; }
```

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
typedef int T;
typedef struct Queue *Link_Queue;
inline void ERR(const char* s) { fputs(s,stderr);  exit(1); }

struct Queue { int NumEl, Size, P; T Items[]; };

Link_Queue Queue_ini(int N) { Link_Queue a = (Link_Queue)malloc(sizeof(struct Queue) + N*sizeof(T));
                              a->Size=N; a->NumEl=a->P=0; return a; }
Link_Queue Queue_free(Link_Queue a) { free(a); return NULL; }
int Queue_Is_Empty(const Link_Queue a) { return a->NumEl==0; }
int Queue_Is_Full(const Link_Queue a) { return a->NumEl==a->Size; }
void Queue_put(Link_Queue a, T New_el) { if(Queue_Is_Full(a)) ERR("Queue::put: queue is full");
                              a->Items[a->P++]=New_el; a->NumEl++; if(a->P==a->Size) a->P=0; }
T Queue_get(Link_Queue a) { if(Queue_Is_Empty(a)) ERR("Queue::get: queue is empty");
                              return a->Items[a->P - a->NumEl-- + (a->P<=a->NumEl?a->Size:0)]; }

int main() {
int k,N;
    scanf("%d",&N);
Link_Queue a = Queue_ini(N);
    while(scanf("%d",&k)!=EOF) Queue_put(a,k);
    while(!Queue_Is_Empty(a)) { k=Queue_get(a); printf("%d ",k); }
    a=Queue_free(a);
    return 0;
}
```

запоминаем тип

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
typedef int T;
typedef struct Que
inline void ERR(cc

struct Queue { int

Link_Queue Queue_i
```

```
5
1 2 3 4 5 6
Queue::put: queue is full
```

```
Link_Queue Queue_free(Link_Queue a) { free(a); return NULL; }
int Queue_Is_Empty(const Link_Queue a) { return a->NumEl==0; }
int Queue_Is_Full(const Link_Queue a) { return a->NumEl==a->Size; }
void Queue_put(Link_Queue a, T New_el) { if(Queue_Is_Full(a)) ERR("Queue::put: queue is full");
                                a->Items[a->P++]=New el; a->NumEl++; if(a->P==a->Size) a->P=0; }
T Queue_get(Link_Queue a) { if(Queue_Is_Er
                                return a->It

int main() {
int k,N;
    scanf("%d",&N);
Link_Queue a = Queue_ini(N);
    while(scanf("%d",&k)!=EOF) Queue_put(a
    while(!Queue_Is_Empty(a)) { k=Queue_ge
    a=Queue_free(a);
    return 0;
}
```

```
10
1 2 3 4 5 6 7 8 9 0
^Z
1 2 3 4 5 6 7 8 9 0
```

```c
#include <stdio.h>
typedef int T;

#include "QS.h"

int main() {
int k,N; scanf("%d",&N);

Link_Queue a = Queue_ini(N);

    for(k=0;k<N;k++) Queue_put(a,k);
    for(k=0;k<(N+1)/2;k++) Queue_get(a);
    for(k=0;k<(N+1)/2;k++) Queue_put(a,N+k);
    while(!Queue_Is_Empty(a)){ k=Queue_get(a); printf("%d ",k); }
    a=Queue_free(a);

    return 0;
}
```

спрятали реализацию

D:\A1\Lect2015\Exmp\9\mas_ë\t3.exe

```
10
5 6 7 8 9 10 11 12 13 14
--------------------------------
```

```
#include <string.h>
typedef char* T;
#include "QS.h"

char* fget_str(char *s, FILE* a) { const int N=10;
char c[N]; int b=1; int ls=0,lc; if(fgets(c,N,a)==NULL) return NULL;
    s = (char*)calloc(1,1);
    do { lc=strlen(c); if(c[lc-1]=='\n') { b=0; c[--lc]='\0'; }
        s = (char*)realloc(s,(ls+=lc)+1);
        strcat(s,c); } while(b && fgets(c,N,a));
    return s;
}

int main() {
Link_Stack a;
Link_Queue b;
char* k;

    a = Stack_ini(100);
    b = Queue_ini(100);
    while(k=fget_str(k,stdin)) { Stack_push(a,k); Queue_put(b,k); }
    while(!Queue_Is_Empty(b)) { k=Queue_get(b); puts(k); }
    b = Queue_free(b);
    while(!Stack_Is_Empty(a)) { k=Stack_pop(a); puts(k); }
    a = Stack_free(a);

    return 0;
}
```

вспоминаем тип

```c
#include <stdio.h>

struct complex { double Re; double Im; };
void print_complex(const struct complex *a) { printf("(%lf,%lf) ",a->Re,a->Im); };

typedef struct complex T;

#include "QS.h"

int main() {
Link_Stack a;
Link_Queue b;
T k;
    a = Stack_ini(100);
    b = Queue_ini(100);
    while(scanf("%lf%lf",&k.Re,&k.Im)!=EOF) { Stack_push(a,k); Queue_put(b,k); }
    while(!Stack_Is_Empty(a)) { k=Stack_pop(a); print_complex(&k); }
    a = Stack_free(a); puts("");
    while(!Queue_Is_Empty(b)) { k=Queue_get(b); print_complex(&k); }
    b = Queue_free(b);

    return 0;
}
```

вспоминаем тип

```
1 2 3 4 5 6
^Z
(5.000000,6.000000) (3.000000,4.000000) (1.000000,2.000000)
(1.000000,2.000000) (3.000000,4.000000) (5.000000,6.000000)
--------------------------------
Process exited after 7.953 seconds with return value 0
Для продолжения нажмите любую клавишу . . .
```

РЕАЛИЗАЦИЯ СТРУКТУРАМИ

Компоненты

```
#include <stdlib.h>
#include <malloc.h>

typedef struct item *Link_Item;
typedef struct list *Link_List;
typedef struct stack *Link_Stack;
typedef struct queue *Link_Queue;

void ERR(const char* s) {
     fputs(s,stderr); exit (1); }
```

Узел. Конструкция

```
typedef struct item { T node; Link_Item next; }
     Item;

Link_Item Item_Create(T elem, Link_Item n) {
     Link_Item rab =
          (Link_Item)calloc(1,sizeof(Item));
     rab->node=elem; rab->next=n; return rab; }

void Item_Delete(Link_Item a) { free(a); }
```

Узел. Действия

```
T Item_get_node(const Link_Item a) {
     return a->node; }

Link_Item Item_get_next(Link_Item a) {
     return a->next; }
```

Стек. Конструкция и состояние

```
typedef struct stack { Link_Item Top; } Stack;

Link_Stack Stack_ini(Link_Stack a) {
     return a =
          (Link_Stack)calloc(1,sizeof(Stack)); }

Link_Stack Stack_Delete(Link_Stack a) {
     free(a); return NULL; }

int Stack_Is_Empty(const Link_Stack a) {
     return a->Top==NULL; }
```

Стек. Действия

```
void Stack_push(Link_Stack a, T elem) {
     a->Top=Item_Create(elem,a->Top); }

T Stack_pop(Link_Stack a) {
     if(Stack_Is_Empty(a))
          ERR("Stack::pop: empty stack");
     Link_Item p = a->Top;
     T rab = Item_get_node(p);
     a->Top = Item_get_next(p);
     Item_Delete(p);
     return rab;}
```

Очередь. Конструкция и состояние

```
typedef struct queue {
     Link_Item Head; Link_Item Tail; } Queue;

Link_Queue Queue_ini(Link_Queue a) { return a =
     (Link_Queue)calloc(1,sizeof(struct Queue));}

Link_Queue Queue_Delete(Link_Queue a) {
     free(a); return NULL; }

int Queue_Is_Empty(const Link_Queue a) {
     return a->Head==NULL; }
```

Очередь. Действия

```
void Queue_put(Link_Queue a, T elem) {
  if(Queue_Is_Empty(a))
    a->Tail=a->Head=Item_Create(elem,NULL);
  else a->Tail =
    (a->Tail->next = Item_Create(elem,NULL)); }

T Queue_get(Link_Queue a) {
  if(Queue_Is_Empty(a))
    ERR("Queue::get: gueue is empty");
  Link_Item p = a->Head;
  T rab = Item_get_node(p);
  a->Head = Item_get_next(p);
  Item_Delete(p);
  if(Queue_Is_Empty(a)) a->Tail=NULL;
  return rab; }
```

Список. Конструкция и состояние

```
typedef struct list {
  Link_Item Front;
  Link_Item Back;
} List;

Link_List List_ini(Link_List a) {
  return a = (Link_List)calloc(1,sizeof(List));}

int List_Is_Empty(Link_List a) {
  return a->Front==NULL; }
```

Список. Операции. Поиск

```
Link_Item List_Find(
    Link_List a,
    Link_Item *F,
    T key ) {
  if(List_Is_Empty(a)) return (*F=NULL);
  Link_Item ptr = *F = a->Front;
  if(Item_get_node(*F)==key) return NULL;
  while((*F=Item_get_next(ptr))!=NULL) {
    if(Item_get_node(*F)==key) break; ptr=*F; }
  return ptr; }
```

Список. Операции. Добавление 1

```
void List_Insert_front(Link_List a, T elem) {
  a->Front = Item_Create(elem,a->Front);
  if(a->Back==NULL) a->Back = a->Front; }

int List_Insert_back(Link_List a, T elem) {
  if(List_Is_Empty(a))
    a->Front = a->Back = Item_Create(elem,NULL);
  else
    a->Back =(a->Back->next =
             Item_Create(elem,NULL)); }
```

Список. Операции. Добавление 2

```
int List_Insert_after(
     Link_List a,
     T elem,
     T after) {
  Link_Item c;
  List_Find(a,&c,after);
  if(c==NULL) return 0;
  c->next=Item_Create(elem,Item_get_next(c));
  return 1; }
```

Список. Операции. Удаление 1

```
T List_Remove_front(Link_List a) {
  if(List_Is_Empty(a))
    ERR("List::Remove_front: list is empty");
  Link_Item p=a->Front; T rab=Item_get_node(p);
  a->Front=Item_get_next(p); Item_Delete(p);
  if(a->Front==NULL) a->Back=NULL; return rab; }

int List_Remove(Link_List a, T key) {
  Link_Item b; Link_Item c;
  b=List_Find(a,&c,key);
  if(c==NULL) return 0;
  if(b==NULL) a->Front=Item_get_next(a->Front);
  else b->next=Item_get_next(c);
  Item_Delete(c); return 1; }
```

Список. Операции. Удаление 2

```
T List_Remove_back(Link_List a) {
  if(List_Is_Empty(a))
    ERR("List::Remove_back: list is empty");
  Link_Item p=a->Front;
  T rab = Item_get_node(a->Back);
  if(a->Front==a->Back)
    a->Front = a->Back = NULL;
  else { while(p->next!=a->Back) p=p->next;
    a->Back=p; p=p->next; a->Back->next=NULL; }
  Item_Delete(p); return rab; }
```

Список. Операции. Разворот

```
void List_Revers(Link_List a) {
  Link_Item p=NULL;
  while(a->Front!=NULL)
    p=Item_Create(List_Remove_front(a),p);
  a->Front=p;
  while(Item_get_next(p)!=NULL)
    p=Item_get_next(p);
  a->Back=p; }
```

Список. Операции. Сортировка

```
void List_Sort (
    Link_List a,
    int (*f)(const T a, const T b)) {
  Link_Item F=0;
  while(a->Front!=NULL){
    Link_Item p = a->Front;
    T rab = Item_get_node(p);
    while(p=Item_get_next(p))
      if(f(p->node,rab)>0)rab=Item_get_node(p);
    F=Item_Create(rab,F); List_Remove(a,rab); }
  a->Front = F;
  while(F->next) F=F->next;
  a->Back=F;}
```

SQL.H (1)

```c
#include <stdlib.h>
#include <malloc.h>

typedef struct item *Link_Item;
typedef struct list *Link_List;
typedef struct stack *Link_Stack;
typedef struct queue *Link_Queue;

inline void ERR(const char* s) { fputs(s,stderr); exit (1); }

typedef struct item { T node; Link_Item next; } Item;
Link_Item Item_Create(const T elem, const Link_Item n) {
    Link_Item rab = (Link_Item)calloc(1,sizeof(Item));
    rab->node=(T)elem; rab->next=n; return rab; }
T Item_get_node(const Link_Item a) { return a->node; }
Link_Item Item_get_next(const Link_Item a) { return a->next; }
void Item_Delete(Link_Item a) { free(a); }

typedef struct stack { Link_Item Top; } Stack;
Link_Stack Stack_ini(Link_Stack a) { return a = (Link_Stack)calloc(1,sizeof(Stack));
int Stack_Is_Empty(const Link_Stack a) { return a->Top==NULL; }
void Stack_push(Link_Stack a, const T elem) { a->Top=Item_Create(elem,a->Top); }
T Stack_pop(Link_Stack a) {
    if(Stack_Is_Empty(a)) ERR("Stack::pop: empty stack");
    Link_Item p = a->Top; T rab = Item_get_node(p); a->Top = Item_get_next(p);
    Item_Delete(p); return rab; }
Link_Stack Stack_Delete(Link_Stack a) { free(a); return NULL; }
```

struct Stack - SQL.h (19) - Ctrl+Click for more info

SQL.H (2)

```
typedef struct queue { Link_Item Head; Link_Item Tail; } Queue;
Link_Queue Queue_ini(Link_Queue a) { return a = (Link_Queue)calloc(1,sizeof(Queue)); }
int Queue_Is_Empty(const Link_Queue a) { return a->Head==NULL; }
void Queue_put(Link_Queue a, const T elem) {
    if(Queue_Is_Empty(a)) a->Tail = a->Head = Item_Create(elem,NULL);
    else a->Tail= (a->Tail->next = Item_Create(elem,NULL)); }
T Queue_get(Link_Queue a) {
    if(Queue_Is_Empty(a)) ERR("Queue::get: gueue is empty");
    Link_Item p = a->Head; T rab = Item_get_node(p); a->Head=Item_get_next(p);
    Item_Delete(p); if(Queue_Is_Empty(a)) a->Tail=NULL; return rab; }
Link_Queue Queue_Delete(Link_Queue a) { free(a); return NULL; }
```

```
typedef struct list { Link_Item Front; Link_Item Back; } List;
Link_List List_ini(Link_List a) { return a = (Link_List)calloc(1,sizeof(List)); }
int List_Is_Empty(const Link_List a) { return a->Front==NULL; }
Link_Item List_Find(const Link_List a, Link_Item *F, const T key) { if(List_Is_Empty(a)) return (*F=NULL);
    Link_Item ptr = *F = a->Front; if(Item_get_node(*F)==key) return NULL;
    while((*F=Item_get_next(ptr))!=NULL) { if(Item_get_node(*F)==key) break; ptr=*F; }
    return ptr; }
void List_Insert_front(Link_List a, const T elem) { a->Front = Item_Create(elem,a->Front);
    if(a->Back==NULL) a->Back = a->Front; }
int List_Insert_after(Link_List a, const T elem, const T after) { Link_Item c; List_Find(a,&c,after);
    if(c==NULL) return 0; c->next = Item_Create(elem,Item_get_next(c)); return 1; }
void List_Insert_back(Link_List a, const T elem) {
    if(List_Is_Empty(a)) a->Front = a->Back = Item_Create(elem,NULL);
    else a->Back = (a->Back->next = Item_Create(elem,NULL)); }
```

```
T List_Remove_front(Link_List a) { if(List_Is_Empty(a)) ERR("List::Remove_front: list is empty");
    Link_Item p=a->Front; T rab=Item_get_node(p); a->Front=Item_get_next(p); Item_Delete(p);
    if(a->Front==NULL) a->Back=NULL; return rab; }
int List_Remove(Link_List a, const T key) { Link_Item b; Link_Item c; b=List_Find(a,&c,key); if(c==NULL) return 0;
    if(b==NULL) a->Front=Item_get_next(a->Front); else b->next=Item_get_next(c); Item_Delete(c);
    return 1; }
T List_Remove_back(Link_List a) { if(List_Is_Empty(a)) ERR("List::Remove_back: list is empty");
    Link_Item p=a->Front; T rab = Item_get_node(a->Back);
    if(a->Front==a->Back) a->Front = a->Back = NULL;
    else { while(p->next!=a->Back) p=p->next; a->Back=p; p=p->next; a->Back->next=NULL; }
    Item_Delete(p); return rab; }
void List_Revers(Link_List a) { Link_Item p=NULL; while(a->Front!=NULL) p=Item_Create(List_Remove_front(a),p);
    a->Front=p; while(Item_get_next(p)!=NULL) p=Item_get_next(p); a->Back=p; }
void List_Sort(Link_List a, int (*f)(const T, const T)) { Link_Item F=NULL;
    while(a->Front!=NULL) { Link_Item p = a->Front; T rab = Item_get_node(p);
        while(p=Item_get_next(p)) if(f(p->node,rab)>0) rab=Item_get_node(p);
        F=Item_Create(rab,F); List_Remove(a,rab); }
    a->Front = F; while(F->next) F=F->next; a->Back=F; }
Link_List List_Delete(Link_List a) { free(a); return NULL; }
```

SQL.H (3)

запоминаем тип

t_int.c (1)

```c
#include <stdio.h>
#include <string.h>
#include <windows.h>

typedef int T;

#include "SQL.h"

void List_print(Link_List a) { Link_Item p = a->Front; if(!p) return;
    do printf("%d ",Item_get_node(p)); while(p=Item_get_next(p)); puts(""); }

int cmp(T a, T b) { if(a>b) return 1; else if(a<b) return -1; else return 0; }

int main() {
Link_Stack a;
Link_Queue b;
Link_List c;

int k;

HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

    a = Stack_ini(a);
    b = Queue_ini(b);
    c = List_ini(c);
    while(scanf("%d",&k)!=EOF) { List_Insert_back(c,k); Stack_push(a,k); Queue_put(b,k); }
```

t_int.c (2)

```
SetConsoleTextAttribute(hStdOut,FOREGROUND_BLUE | FOREGROUND_INTENSITY);
    while(!Stack_Is_Empty(a)) { k=Stack_pop(a); printf("%d ",k); } puts("");
    a = Stack_Delete(a);

SetConsoleTextAttribute(hStdOut,FOREGROUND_GREEN | FOREGROUND_INTENSITY);
    while(!Queue_Is_Empty(b)) { k=Queue_get(b); printf("%d ",k); } puts("");
    b = Queue_Delete(b);

SetConsoleTextAttribute(hStdOut,FOREGROUND_RED | FOREGROUND_INTENSITY);
    List_Insert_after(c,-1,5);
    List_Remove(c,4);
    List_Remove(c,1);
    List_Remove(c,6);
    List_print(c);

SetConsoleTextAttribute(hStdOut,FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_INTENSITY);
    List_Revers(c);
    List_Insert_back(c,7);
    List_print(c);

SetConsoleTextAttribute(hStdOut,FOREGROUND_GREEN | FOREGROUND_BLUE | FOREGROUND_INTENSITY);
    List_Sort(c,cmp);
    List_print(c);
    c = List_Delete(c);

    return 0;
}
```

```c
SetConsoleTextAttribute(hStdOut,FOREGROUND_BLUE | FOREGROUND_INTENSITY);
    while(!Stack_Is_Empty(a)) { k=Stack_pop(a); printf("%d ",k); } puts("");
    a = Stack_Delete(a);

SetConsoleTextAttribute(hStdOut,FOREGROUND_GREEN | FOREGROUND_INTENSITY);
    while(!Queue_Is_Empty(b)) { k=Queue_get(b); printf("%d ",k); } puts("");
    b = Queue_Delete(b);

SetConsoleTextAttribute(hStdOut,FOREGROUND_RED | FOREGROUND_INTENSITY);
    List_Insert_after(c,-1,5);
    List_Remove(c,4);
    List_Remove(c,1);
    List_Remove(c,6);
    List_print(c);

SetConsoleTextAttribute(hStdOut,FOREGROUND_GREEN | FOREGROUND_RED |
    List_Revers(c);
    List_Insert_back(c,7);
    List_print(c);

SetConsoleTextAttribute(hStdOut,FOREGROUND_GREEN | FOREGROUND_BLUE |
    List_Sort(c,cmp);
    List_print(c);
    c = List_Delete(c);

    return 0;
}
```

c (3)

t_str.c (1)

```c
#include <stdio.h>
#include <string.h>
#include <windows.h>

#define T char*

#include "SQL.h"

void List_print(Link_List a) { Link_Item p = a->Front; if(!p) return;
    do printf("%s\n",Item_get_node(p)); while(p=Item_get_next(p)); }

char* fget_str(char *s,FILE* a) { const int N=10;
char c[N]; int b=1; int ls=0,lc;
    if(fgets(c,N,a)==NULL) return NULL;
    s = (char*)calloc(1,1);
    do { lc=strlen(c); if(c[lc-1]=='\n') { b=0; c[--lc]='\0'; }
        s = (char*)realloc(s,(ls+=lc)+1); strcat(s,c);
    } while(b && fgets(c,N,a));
    return s;
}

int main() {
Link_Stack a;
Link_Queue b;
Link_List c;

T k;
```

вспоминаем тип

D:\A1\Lect2015\Exmp\9\strc_c\t_char.c - Dev-C++ 5.10

Файл Правка Поиск Вид Проект Выполнить Сервис AStyle Окно Справка

TDM-GCC 4.8.1 32-bit Debug

(globals)

t_char.c SQL.h

t_str.c (2)

```
29  HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);
30
31      a = Stack_ini(a);
32      b = Queue_ini(b);
33      c = List_ini(c);
34      while(k=fget_str(k,stdin)) { List_Insert_back(c,k); Stack_push(a,k); Queue_put(b,k); }
35
36  SetConsoleTextAttribute(hStdOut,FOREGROUND_BLUE | FOREGROUND_INTENSITY);
37      while(!Stack_Is_Empty(a)) { k=Stack_pop(a); printf("%s\n",k); }
38      a = Stack_Delete(a);
39
40  SetConsoleTextAttribute(hStdOut,FOREGROUND_GREEN | FOREGROUND_INTENSITY);
41      while(!Queue_Is_Empty(b)) { k=Queue_get(b); printf("%s\n",k); }
42      b = Queue_Delete(b);
43
44  SetConsoleTextAttribute(hStdOut,FOREGROUND_RED | FOREGROUND_INTENSITY);
45      List_print(c);
46      List_Revers(c);
47
48  SetConsoleTextAttribute(hStdOut,FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_INTENSITY);
49      List_print(c);
50      List_Sort(c,strcmp);
51
52  SetConsoleTextAttribute(hStdOut,FOREGROUND_BLUE | FOREGROUND_RED | FOREGROUND_INTENSITY);
53      List_print(c);
54      c = List_Delete(c);
55
```

Компилятор | Ресурсы | Журнал компиляции | Отладка | Результаты поиска

Line: 51 Col: 5 Sel: 0 Lines: 57 Length: 1622 Вставка Done parsing in 0,047 seconds

t_str.c (3)

```
D:\A1\Lect2015\Exmp\9\strc_c>t_char < tst.cpp
}
    return 0;
    while( cin >> k ) cout << "k= " << k << ": " << 2.+f(2) << endl;
    cout.precision(12); cout << exp(1.) << endl;
int main() {

double f(int n) { return n > k ? 1. : n/(n+f(n+1)); }
int k;
using namespace std;
#include <cmath>
#include <iostream>
#include <iostream>
#include <cmath>
using namespace std;
int k;
double f(int n) { return n > k ? 1. : n/(n+f(n+1)); }

int main() {
    cout.precision(12); cout << exp(1.) << endl;
    while( cin >> k ) cout << "k= " << k << ": " << 2.+f(2) << endl;
    return 0;
}
#include <iostream>
#include <cmath>
using namespace std;
int k;
double f(int n) { return n > k ? 1. : n/(n+f(n+1)); }

int main() {
    cout.precision(12); cout << exp(1.) << endl;
    while( cin >> k ) cout << "k= " << k << ": " << 2.+f(2) << endl;
    return 0;
}
```

```
}
    return 0;
    while( cin >> k ) cout << "k= " << k << ": " << 2.+f(2) << endl;
    cout.precision(12); cout << exp(1.) << endl;
int main() {

double f(int n) { return n > k ? 1. : n/(n+f(n+1)); }
int k;
using namespace std;
#include <cmath>
#include <iostream>

    cout.precision(12); cout << exp(1.) << endl;
    return 0;
    while( cin >> k ) cout << "k= " << k << ": " << 2.+f(2) << endl;
#include <cmath>
#include <iostream>
double f(int n) { return n > k ? 1. : n/(n+f(n+1)); }
int k;
int main() {
using namespace std;
}
```

t_str.c (4)

Ещё понятия

- Полиморфизм
- Шаблон функции
- Инкапсуляция
- Класс
- Шаблон класса

Полиморфизм

разные функции с одинаковым именем и различными параметрами

```
int abs(int a) { return a>=0?a:-a; }
double abs(double a) { return a>=0?a:-a; }
```

Шаблоны функций

```
template<typename T> T abs(T a) {
                return a>=0?a:-a; }

int x; …   y = abs(x); // int abs(int);

float u; … v = abs(u); // float abs(float);
```

Инкапсуляция

```
complex a, b(1.,2.);
```

```
struct complex { float Re,Im;
  complex() { Re = Im = 0; }
  complex(float R, float I) { Re=R; Im=I; }
  float abs() { return sqrt(Re*Re + Im*Im); }
  complex operator+(complex a) {
    return complex(Re+a.Re, Im+a.Im);}
};

ostream& operator<<(
    ostream& a,const complex& b){
  return a << '(' << b.Re << ',' << b.Im << ')'
 << endl; }
```

```
a.abs();
a.operator+(b); // a+b;
```

Класс

```
class complex { float Re,Im;
public:
complex() { Re = Im = 0; }
complex(float R, float I) { Re=R; Im=I; }
float abs() { return sqrt(Re*Re + Im*Im); }
complex operator+(complex a) {
  return complex(Re+a.Re, Im+a.Im); }

friend ostream& operator<<(
    ostream& a, const complex& b) {
   return a << '(' << b.Re << ',' << b.Im << ')'
  << endl; }
};
```