Указатели на функции (УФ)

Термины, имеющие отношение к УФ:

- Адрес функции
- Функция как формальный параметр, процедурный тип (Паскаль)
- Динамическое связывание
- Виртуальные функции (полиморфизм) – ООП
- Лямбда-выражения, анонимные функции
- Поток – функция, исполняемая параллельно (асинхронно) к main

Архитектурно-зависимые идеи:

- Функция во внутреннем представлении – участок сегмента кода (CS), содержащий программный код (команды)
- Во внутреннее представление компилируется тело, заголовок – описание интерфейса, в каком виде тело получает в стеке формальные параметры
- Указатель на функцию – начальный адрес тела функции
- Указатель на функцию можно передавать в качестве параметра, выполнять тело (вызывать функцию) = функция как ОБЪЕКТ, над которым можно производить указанные действия
- Функциональное программирование (парадигма) – последовательность
- Императивное программирование – описание процесса исполнения последовательности команд, изменяющих данные

Указатели на функции (УФ)

Синтаксис:

- определение в контексте использования
- УФ = адрес + интерфейс (прототип)

```
int     (*pf)();
int     (*pf)(void);
int     (*pf)(int, char*);
```

```
int     INC(int a) { return a+1; }
extern  int  DEC(int);
int     (*pf)(int);
pf = &INC;
pf = INC;                          // присваивание указателя
int     (*pp)(int) = &DEC;         // инициализация указателя
```

Аналогия с массивом: имя функции без скобок = адрес

Преобразования int = маш. слово = УФ

```
void (*pf)(void) = (void(*)(void))0x1000;   // Константа 1000 – шестнадцатеричный адрес
void main() { (*pf)(); }                    // функции, которую вызывает main
                                            // void(*)(void) –абстрактный тип данных –
                                            // указатель на функцию
```

Указатели на функции (УФ)

Связывание = установление соответствия между свойствами объекта программы (тип, переменная, функция) и элементами архитектуры (адрес, значение, команда)

Время связывания = этап трансляции/исполнения – определение языка (1), реализация компилятора (2), компиляция (3), компоновка (4), загрузка (5), исполнение (6, runtime)

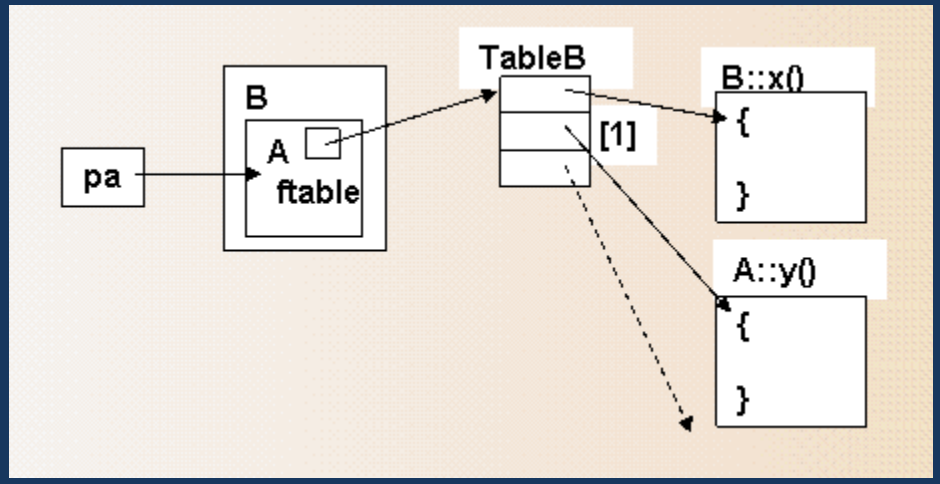
Раннее (статическое) связывание – 1-4, позднее (динамическое) – 5,6

Статические массивы в Си, размерность – 3, динамические – 6

Обычные функции: связывание «имя – адрес» - 3,4

Указатели на функции (УФ)

Динамические связывание – виртуальные функции = создание объекта (переменной)



При создании объекта в его базовый класс записывается адрес таблицы функций для окружающего его производного класса

Указатели на функции (УФ)

Таблица функций: вызов функции «по имени»

```
#define PI 3.1415926
char    *names[] = { "sin","cos","tan",NULL};       // Массив имен (указатели на строки)
double  (*pf[])(double) = { sin, cos, tan};         // Массив функций (адреса функций)
//-----------------------------------------------------93-01.cpp
//---- Вызов функции по имени из заданного списка
 double call_by_name(char *pn, double arg) {
 for ( int i=0; names[i]!=NULL; i++)
       if (strcmp(names[i],pn) == 0) {   // Имя найдено -
      return ((*pf[i])(arg));            // вызов функции по i-му
      }                                  // указателю в массиве pf
 return 0.;}

 void main(){
 printf("cos(pi/4)=%lf\n",call_by_name("cos",3.1415926/4));}
```

Численное интегрирование: подынтегральная функция = формальный параметр

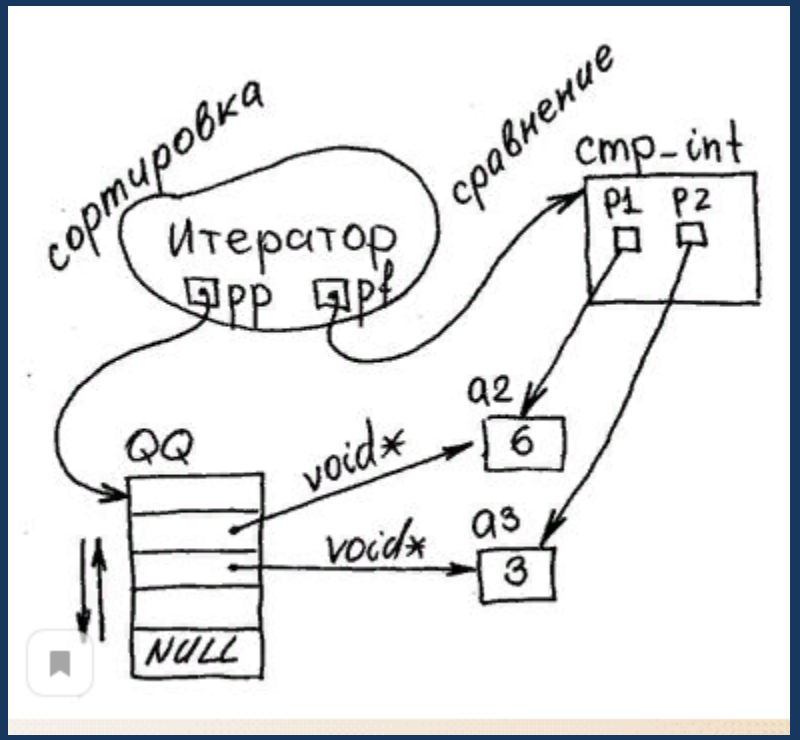
```
//-----------------------------------------------------93-02.cpp
//------Численное интегрирование произвольной функции
// a,b - границы интегрирования, n - число точек
// pf - подынтегральная функция
 double INTEG(double a, double b, int n, double(*pf)(double))
 { double s,h,x;
 for (s=0., x=a, h = (b-a)/n; x <=b; x+=h) s += (*pf)(x) * h;
 return s; }
 extern double sin(double);
 void main() { printf("sin(0..pi/2)=%lf\n",INTEG(0.,PI/2,1000,sin)); }
```

Указатели на функции (УФ)

Функция – формальный параметр функции:

- Лямбда – выражение = фактический параметр – тело функции, создаваемой «по месту применения»
- «довесок», исполняющий часть основного алгоритма
  - Основная функция знает, как сортировать, но не знает, как сравнить
  - Функция – параметр знает, как сравнить, но не знает, где взять (что сравнивать)
- Массив структур, способ сравнения – внешняя функция

```
struct user{
    char name[20];
    int account;
    double time;
}   S[]={
    {"John",2775,13.74},
    {"Alex",4765,10.1},
    {"Martin",1122,0.0},
    {"Billy",3344,7.5},
    {"Larry",1222,5.5},
    {"",0,0}};
```



Указатели на функции (УФ)

```
//-------- Вставка погружением
// cmp - указатель на функцию сравнения двух struct user
void sort(user A[], int (*cmp)(user&,user&)){
    for (int i=1;A[i].name[0]!=0;i++)
        for(int j=i; j>0 && (*cmp)(A[j],A[j-1])<0; j--)
        { user c=A[j];
        A[j]=A[j-1];
        A[j-1]=c; }
    }
//--------- функция сравнения по имени
int cmp_name(user &u1, user &u2){
    return strcmp(u1.name,u2.name);
    }
//--------- функция сравнения по account
int cmp_account(user &u1, user &u2){
    return u1.account - u2.account;
    }
//--------- функция сравнения по time
int cmp_time(user &u1, user &u2){
    return u1.time - u2.time;
    }
```

```
void main(){
    show(S);
    sort(S,cmp_name);
    show(S);
    sort(S,cmp_account);
    show(S);
    sort(S,cmp_time);
    show(S);
    }
```

Предельный случай: переменные произвольного типа, задаваемые указателями void* - «чистый» адрес

Указатели на функции (УФ)

Головоломки

```
//-----------------------------------------
void ( *P1(void(*ff)(void)))(void) {
    return ff; }
void foo1(void){ printf("I'm foo\n"); }
void main1(){
    (*P1(foo1))();
    }
//-----------------------------------------
```

main1 – вызывает P1, передает ей адрес foo1, получает его обратно и вызывает ее по этому адресу

P1 – получает адрес функции (ff) и возвращает обратно в ax

Disassembly

Function: P1 (D:\Temp\fufu\main.cpp:7)
Frame start: 0x22ff20

```
0x401460    push   %ebp
0x401461    mov    %esp,%ebp
0x401463    mov    0x8(%ebp),%eax
0x401466    pop    %ebp
0x401467    ret
```

Mixed Mode | Adjust | Save to text file

Disassembly

Function: main1 (D:\Temp\fufu\main.cpp:10)
Frame start: 0x22ff40

```
0x40147e    mov    %esp,%ebp
0x401480    sub    $0x18,%esp
0x401483    movl   $0x401468,(%esp)
0x40148a    call   0x401460 <P1(void (*)())>
0x40148f    call   *%eax
0x401491    nop
0x401492    leave
0x401493    ret
```

Mixed Mode | Adjust | Save to text file

Вызвать по адресу в ax

Код функции в ДМ

- имя функции = адрес, указатель на функцию
- преобразование char* - адрес «физического представления», массив char
- копирование кода в ДМ
- вызов по указателю, преобразованному в указатель на функцию

```
#include <stdio.h>
int inc1(int vv){
    vv++;
    return vv;
    }
int inc2(int vv){
    vv+=2;
    return vv;
    }
char *copyCode(char *p1, char *p2){
    int sz = p2-p1;
    printf("Code size=%d\n",sz);
    char *out = new char[sz];
    for(int i=0;i<sz;i++)
        out[i]=p1[i];
    return out;
    }
int main(){
    char *q = copyCode((char*)inc1,(char*)inc2);
    int (*pf)(int) = (int(*)(int)) q;
    printf("inc1=%x inc2=%x pf=%x q=%x\n",inc1,inc2,pf,q);
    int v2 = (*pf)(3);
    printf("Result=%d\n",v2);
    v2 = ((int(*)(int))copyCode((char*)inc1,(char*)inc2))(5);
    printf("Result=%d\n",v2);
    getchar();
    }
```

```
D:\temp\fufu\bin\Debug\fufu.exe
Code size=12
inc1=401460 inc2=40146c pf=521550 q=521550
Result=4
Code size=12
Result=6
```

int (*pf)(int) – указатель на функцию с прототипом int …(int)

(int (*)(int)) – преобразование в адрес функции (указатель)

Код функции в ДМ

```
int inc1(int vv){
    vv++;
    return vv;
    }
```

Стек:
[8] – vv
[4] – IP (точка возврата)
[0] – BP (копия SP)
eax – результат

```
Frame start: 0x22ff10
;2  :   int inc1(int vv){
0x401460        push   %ebp
0x401461        mov    %esp,%ebp
;3  :           vv++;
0x401463        addl   $0x1,0x8(%ebp)
;4  :           return vv;
0x401467        mov    0x8(%ebp),%eax
;5  :           }
0x40146a        pop    %ebp
0x40146b        ret
```

```
printf("Result=%d\n",inc1(7));
getchar();
}
```

```
;26 :           printf("Result=%d\n",inc1(7));
0x401535        movl   $0x7,(%esp)
0x40153c        call   0x401460 <inc1(int)>
0x401541        mov    %eax,0x4(%esp)
0x401545        movl   $0x40508e,(%esp)
0x40154c        call   0x403b20 <printf>
;27 :           getchar();
0x401551        call   0x403c48 <getchar()>
;28 :           }
0x401556        mov    $0x0,%eax
0x40155b        leave
0x40155c        ret
```

```
--- d:\temp\fufu3\fufu.cpp ---------------
#include <stdio.h>
int inc1(int vv){
00CF13E0  push        ebp
00CF13E1  mov         ebp,esp
00CF13E3  sub         esp,0C0h
00CF13E9  push        ebx
00CF13EA  push        esi
00CF13EB  push        edi
00CF13EC  lea         edi,[ebp-0C0h]
00CF13F2  mov         ecx,30h
00CF13F7  mov         eax,0CCCCCCCCh
00CF13FC  rep stos    dword ptr es:[edi]
    vv++;
00CF13FE  mov         eax,dword ptr [vv]
00CF1401  add         eax,1
00CF1404  mov         dword ptr [vv],eax
    return vv;
00CF1407  mov         eax,dword ptr [vv]
    }
00CF140A  pop         edi
00CF140B  pop         esi
00CF140C  pop         ebx
00CF140D  mov         esp,ebp
00CF140F  pop         ebp
00CF1410  ret
```

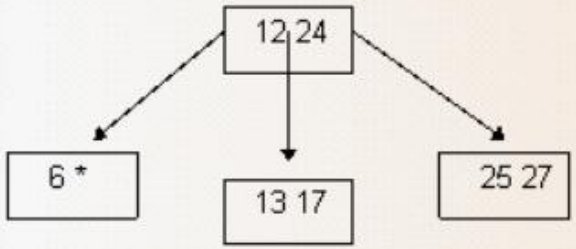
```
D:\temp\fufu3\Debug\fufu3.exe
Code size=60
inc1=cf10dc inc2=cf1118 pf=5242c8 q=5242c8
```

CodeBlocks – отладчик запрещает вызов функции в области ДП (вне сегмента кода), release – проходит
VS2010 – не разрешает

Комментарии к лаб. работе 9.3

Преобразовать один из вариантов сортировки из предыдущих л.р. с использованием массивов , списков (**6.3**), деревьев (**8.4, 8.5**) в итератор. Проверить его работу на двух структурах данных содержащих указатели на различные типы (например, целые и строки). Массив преобразовать в массив указателей.

3.Вершина дерева содержит два целых числа и три указателя на поддеревья. Данные в дереве упорядочены. Написать функцию включения нового значения в дерево с сохранением упорядоченности.



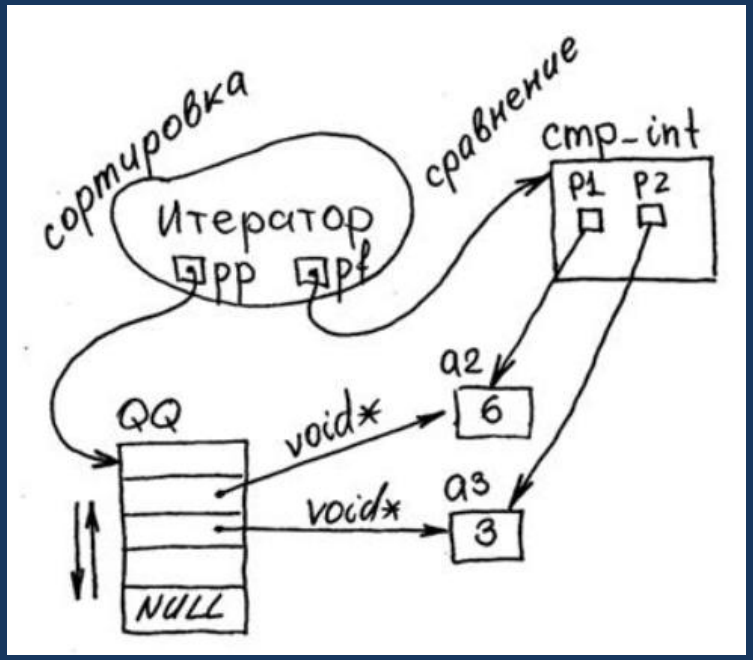
Рефакторинг ???

Комментарии к лаб. работе 9.3

Замена хранимого типа int на void *

```
typedef struct tree {
    struct tree* left;  //указатель на левую ветку
    struct tree* center;//указатель на центральную ветку
    struct tree* right; //указатель на правую ветку
    int val_l;          //меньшее значение
    int val_r;          //большее значение
    int n;              //количество чисел в  дереве
};
```

```
typedef struct tree {
    struct tree* left;      //указатель на левую ветку
    struct tree* center;    //указатель на центральную ветку
    struct tree* right;     //указатель на правую ветку
    void *val_l;            //меньшее значение - указатель
    void *val_r;            //большее значение - указатель
    int n;                  //количество чисел в  дереве
    };
```



Добавление ФП – указатель на функцию сравнения int (*pcmp)(void*,void*)

```
void add(tree* tr, int val) {// ввод дерева  с сохранением упорядоченности
    if (tr->val_l < val && tr->n == 2 && tr->val_r < val) {//если новое значение
        if (tr->right != NULL) {
            add(tr->right, val);
        }
```

```
// ввод дерева  с сохранением упорядоченности
// указатель на данные + указатель на функцию сравнения
void add(tree* tr, void *val, int(*pcmp)(void*,void*)) {
    if (pcmp(tr->val_l,val)<0 && tr->n == 2 && pcmp(tr->val_r,val)<0) {
        //если новое значение больше значений в ячейках добавляем ответвление в правую ветку
        if (tr->right != NULL) {
```

Комментарии к лаб. работе 9.3

Передача ФП по цепочке вызовов

```
void add(tree* tr, int val) {// ввод дерева  с сохранением
    if (tr->val_l < val && tr->n == 2 && tr->val_r < val) {
        if (tr->right != NULL) {
            add(tr->right, val);
        }
```

```
// ввод дерева  с сохранением упорядоченности
// указатель на данные + указатель на функцию сравнения
void add(tree* tr, void *val, int(*pcmp)(void*,void*)) {
    if (pcmp(tr->val_l,val)<0 && tr->n == 2 && pcmp(tr->va
        //если новое значение больше значений в ячейках до
        if (tr->right != NULL) {
            add(tr->right, val, pcmp);
            }
        else {
            tree* b = new tree;
```

Замена операций сравнения int-int на вызов внешней функции

```
void add(tree* tr, int val) {// ввод дерева  с сохранением у
    if (tr->val_l < val && tr->n == 2 && tr->val_r < val) {/
        if (tr->right != NULL) {
            add(tr->right, val);
        }
        else {
```

```
// ввод дерева  с сохранением упорядоченности
// указатель на данные + указатель на функцию сравнения
void add(tree* tr, void *val, int(*pcmp)(void*,void*)) {
    if (pcmp(tr->val_l,val)<0 && tr->n == 2 && pcmp(tr->val_r,val)<0) {
        //если новое значение больше значений в ячейках добавляем ответвл
        if (tr->right != NULL) {
            add(tr->right, val, pcmp);
            }
        else {
```

Комментарии к лаб. работе 9.3

Для вывода (трассировки) – замена cout << val на вызов внешней функции по указателю void (*fput)(void *)

```
void show(tree* tr) { // Функция вывода не отсортированного дерева
    cout << tr->val_r << endl;// вывод наибольшего значения
    if (tr->left != NULL) {//выводим значения левой ветки
        print(tr->left);
        }
    if (tr->center != NULL) {//выводим значения центральной ветки
        print(tr->center);
        }
```

```
}
void show(tree* tr, void(*fput)(void*)) { // Функция вывода не отсортированного дерева
    fput(tr->val_r);
    cout << endl;// вывод наибольшего значения
    if (tr->left != NULL) {//выводим значения левой ветки
        print(tr->left, fput);
        }
    if (tr->center != NULL) {//выводим значения центральной ветки
        print(tr->center, fput);
        }
```

Комментарии к лаб. работе 9.3

Для каждого конкретного типа хранимых элементов:
Функции сравнения и вывода с параметром void* и «сужением» (Java) до хранимого типа

```
int cmpInt(void* p1, void* p2) {
    return *(int*)p1 - *(int*)p2;
    }
void putInt(void* p1) {
    cout << *(int*)p1;
    }
```

Замена манипуляций со структурой данных новыми вызовами

```
tree my_tree;
my_tree.left = NULL;
my_tree.center = NULL;
my_tree.right = NULL;
my_tree.n = 2; // кол-во значений в дереве согласно заданию.
int a[] = { 2,7,3,4,20,-23,10,-28 };
add_tree_top(my_tree, &a[0], &a[1]); // ввод вершины
// ввод значений дерева с сохранением упорядоченности
for(int i=2;i<8;i++)
    add(&my_tree, &a[i], cmpInt);
cout << "Вывод дерева не упорядоченный" << endl;
show(&my_tree,putInt);
cout << "-------Вершина дерева-----------\n";
print_top_tree(my_tree, putInt);
cout << "-------Центральная часть дерева-----------\n";
print_center(&my_tree, putInt);
cout << "-------Левая часть дерева без вершины и центральной части -----------\n";
print_left(&my_tree, putInt);
cout << "-------Правая часть дерева без вершины и центральной части-----------\n";
print_right(&my_tree, putInt);
```