# Структурированный тип. Система типов данных в ЯП и в Си

«Чем же оно живет? – продолжал я размышлять. И пришел к такому выводу: – **Структурой** . Структура государства такова, что даже при нашем минимуме, который мы ему отдаем, оно еще в состоянии всячески себя укреплять…»
**В,Шукшин. О государстве (рассказ)**

1. Структурированный тип – синтаксис и свойства
2. Иерархия типов и переменных, синтаксические и технологические принципы
3. Система типов данных в Си

# 1. Структурированный тип

**Структура – множество разнотипных переменных с общим именем, доступ к которым осуществляется по имени**

От обратного: структура – не массив:
- разнотипные
- идентифицируются не по номеру, а по внутреннему имени

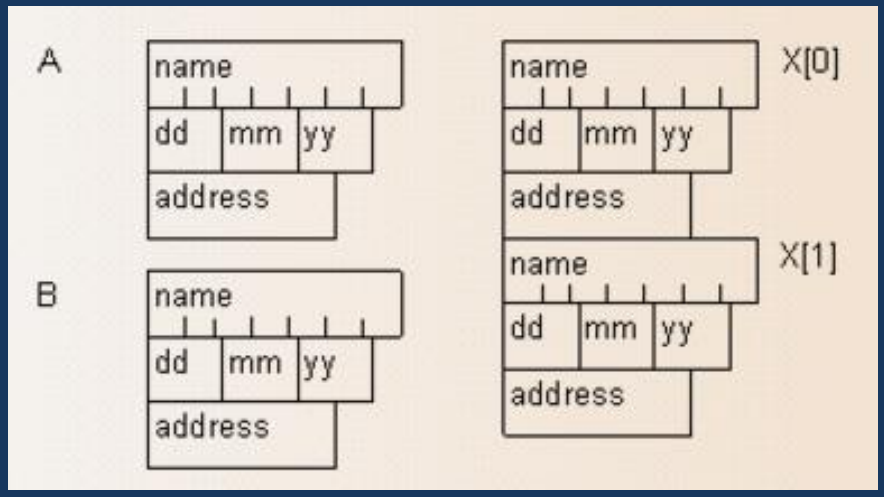
Технологические аспекты:
- структурированная переменная – **объект –** отображение внешней физической или логической (программной) сущности, используемой в программе. **Класс –** описание типа структурированных переменных – основа ООП

Синтаксический аспект:
- Описание ТД отделено от определения переменной

# 1. Синтаксис

- предварительное определение ТД (как такового)
- внутренние – компоненты (поля, записи) – синтаксис псевдо-переменных, все структуры не используются
- компоненты могут интегрироваться внутри или вне (как указатель), во втором случае они существуют отдельно

```
struct man {                    // Определение структурированного типа
        char    name[10];        // Элементы структуры  - поля (имена и типы)
        int     dd,mm,yy;        //
        char    *address;        //
        //-------------------------------------------------------------------------------------
        } A, B, X[10];          // Определение структурированных переменных
```



# 1. Синтаксис

- имя структурированного типа – **ключевое слово** для соответствующего ТД

```
man C,D[20],*p;          // Структурированная переменная C, массив переменных D и
указатель p
man *create()    { ... }   // Функция возвращает указатель на структуру
void  f(man *q)   { ... }   // Функция получает параметр - указатель на структуру
```

- определение и инициализация структурированной переменной

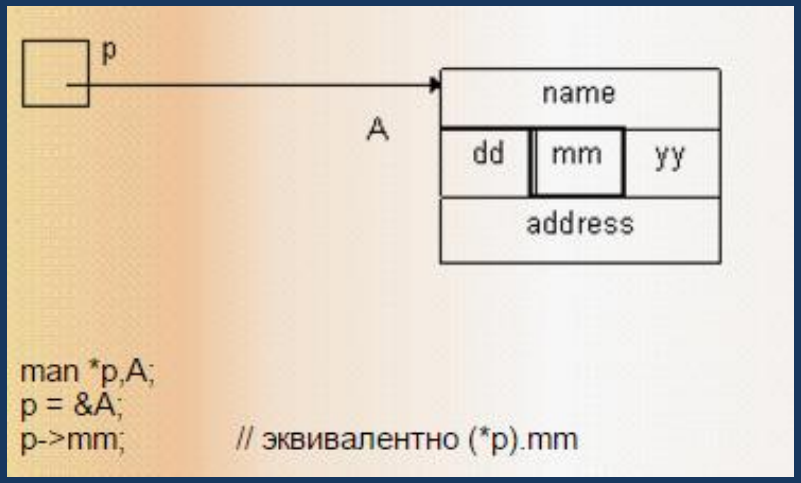
```
man A = { "Петров",1,10,1969,"Морская-12" };
man X[10] =
        { { "Смирнов",12,12,1977,"Дачная-13" },
          { "Иванов" ,21,03,1945,"Северная-21" },
          { ---------------------------------- }
        };
```

- доступ к полю (элементу) структуры – операция «точка»

```
A.name          // элемент name структурированной переменной A
B.dd            // элемент dd структурированной переменной B
```

# 1. Синтаксис

- указатель на структурированный тип – операция **->**
  - стандартный ООП в Си++ базируется на использовании указателя на текущий объект и указателей на динамические объекты (структуры),
  - структуры данных – списки, деревья, последовательности элементов, связанных указателями



# 1. Синтаксис

- **присваивание структур** – побайтное копирование содержимого (массивы - нельзя)
- передача по ссылке, указателю и **по значению (копия) -** технологически объект можно передавать по-разному (конвейер значений или ссылок)

```
void  FF(man X){ …}
void main(){
man A,B[10],*p;
A=B[4];                  // Прямое присваивание структур
p=&A;                    // Присваивание через косвенное обращение по указателю
B[0]=*p;                 // B[0]=A
FF(A); }                 // Присваивание при передаче по значению X=A
```

- **ФП и результат – структура по указателю (ссылке) -** в стеке и на выходе - адрес

```
void proc(man *p){ p->dd++; }      // Для доступа к структуре через указатель
                                   // используется операция ->
void proc1(man &B){ B.dd++; }    // Структура-прототип через ссылку
                                   // доступна «по записи»
void main(){ man A={…,12,5,2001,…};     proc(&A); proc1(A); }
```

# 1. Синтаксис

```
struct man{ …int dd,mm,yy;…};
void proc(man B){                  // Копия структуры – фактического параметра
        x=B.dd;                   // читается, а при изменении не влияет
         B.dd++; }                 // на оригинал (A.dd не меняется)
void main(){
man A={…,12,5,2001,…};
proc(A); }                         // Эквивалент B=A
```

```
struct man{ …int dd,mm,yy;…};          // Эквивалент программы
man proc( man X){                       // man proc(man *out, man X){
X.dd++;                                // X.dd++;
return X;}                             // *out = X;
void main(){                            // man tmp;
man A={…,12,5,2001,…};                  // X=A; out = &tmp;
y=proc(A).dd;                          // выполнить тело proc
}                                      // y=tmp.dd;
```

- **ФП – структура по значению -** в стеке переменная, куда копируется формальный параметр (присваивание структур)
- **результат – структура по значению -** в main создается временная структурированная переменная, функция получает указатель (ссылку), return копирует структурированную переменную из контекста функции по указателю (адресу), переданному параметром

**Резюме:** транслятор поддерживает все манипуляции со структурами как по значению, так и по ссылке (ООП – конструктор копирования для объектов, содержащих связанные через указатели данные, cprog 10.2)

# 2. Иерархия типов и переменных

## Функция, встроенная в struct (метод класса)

- определения struct  заголовок (объявление) или заголовок+тело (определение, inline-подстановка)
- в теле функции работает **контекст структуры (класса).** Имена элементов (полей) структуры, а также других встроенных функций можно использовать непосредственно. Все они имеют отношение к **текущему объекту** (текущей структурированной переменной), с которой в данный момент работает функция
- тело функции может быть включено непосредственно в определение структурированного типа вслед за заголовком (вариант 1) или вынесено за пределы определения структурированного типа. Определение функции (заголовок и тело) дается отдельно, причем в заголовке имя функции присутствует в виде **man::incData –** т.е. включает в себя имя структурированного типа. Этим обозначается, что функция не является самостоятельной, а  принадлежит к этому типу (классу)
- вызов встроенной функции производится только в связке со структурированной переменной в виде **переменная.функция(),** при этом переменная интерпретируется как «текущая», с контекстом которой работает функция

# 2. Иерархия типов и переменных

```
struct man {                          // Определение структурированного типа
        char    name[10];             // Элементы структуры  - поля (имена и типы)
        int     dd,mm,yy;             //
        char    *address;             //
                                      // Вариант 1
        void setData(int d0, int m0, int y0)
                { dd=m0; mm=m0; yy=y0;  }
        void incData();               // Вариант 2
         void plusData(int);          // Вариант 2
         void plusData(int k){        // Вариант 1
                { while(k--!=0) incData(); }
        //-------------------------------------------------------------------------------------------------
        } A, B, X[10];                // Определение структурированных переменных
```

Вынесение тела метода (вариант 2) только в Си++, Java, C# - определение класса одной синтаксической единицей

```
void man::plusData(int k)             // Вариант 2
        { while(k--!=0) incData(); }
void man::incData()                   // Вариант 2
        {  dd++; if (dd==32) dd=1; if (dd==31 && mm…)… }}
void main(){
        A.setData(5,12,1986); A.plusData(16);  }
```

- механизм вызова встроенных функций – неявный указатель **xxx *this**

```
void man_setData(man *this, int d0, int m0, int y0)
                { this->dd=m0; this->mm=m0; this->yy=y0;  }
void man_plusData(man *this, int k)
        { while(this->k--!=0) man_incData(this); }
void man_incData(man *this,)
        {  this->dd++; if (this->dd==32) this->dd=1; if (this->dd==31 && this->mm…)… }}
void main(){
        man_setData(&A,5,12,1986); man_plusData(&A,16);  }
```

# 2. Иерархия типов и переменных

Технологические и синтаксические принципы использования ТД и переменных в программе:

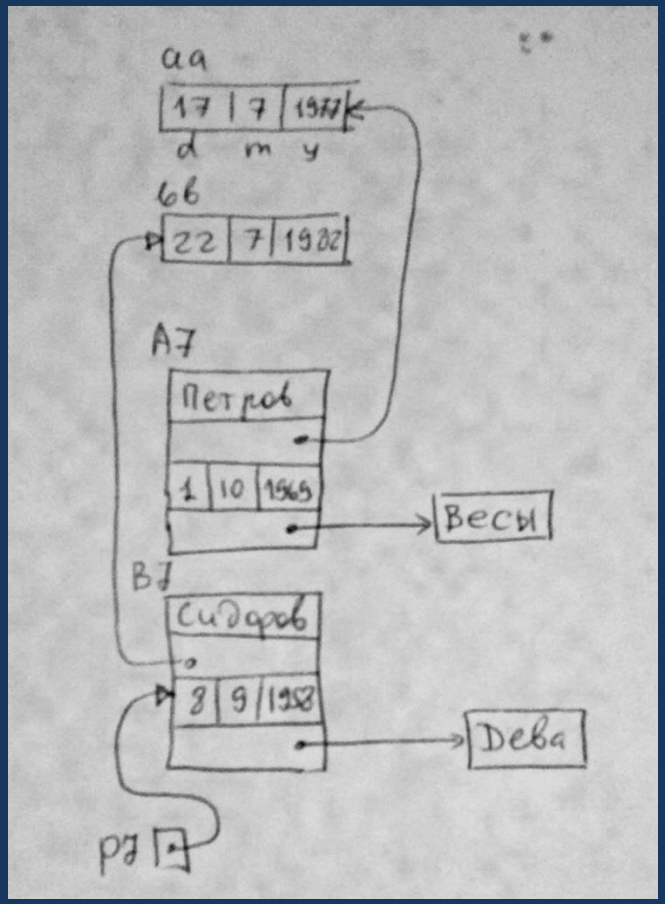
- каждая физическая или логическая сущность, отображаемая м используемая в программе – класс
- сущность может включать другую сущность (вложенность) или ссылаться на нее (указатель)
- представление предметной области в программе является системой связанных объектов (структура данных)
- простейшая структура – иерархическая
- для каждой сущности = структуры разрабатывается набор встроенных функций (методов), выполняющих элементарные действия с ними = методы класса
- модульное проектирование и отладка – отдельно для каждого класса (структурированного типа)
- для обычных компонент (массивы) – вложенность в сам структурированный тип, для указателей – выделение памяти (связанные переменные и динамические массивы) – транслятор выделяет память только под сами указатели

# 2. Иерархия типов и переменных

Примеры связанных структур (инициализация и выражения)

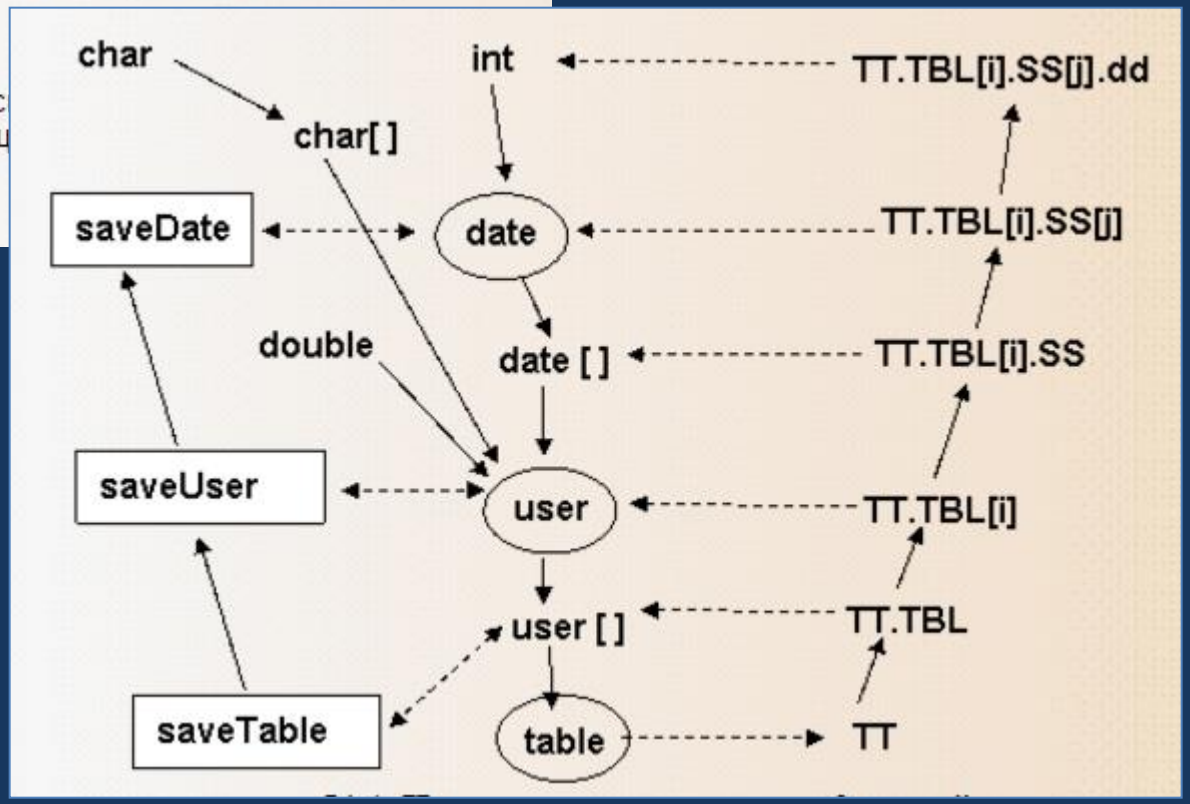
```
//------------------------------------------------------ 6
struct dat7 { int dd,mm,yy; }
aa = { 17,7,1977 },
bb = { 22,7,1982 };
struct man7 {
char name[20];
dat7 *pd;
dat7 dd;
char *zodiak; }
     A7= {"Петров", &aa, { 1,10,1969 }, "Весы" },
     B7= {"Сидоров", &bb, { 8,9,1958 }, "Дева" },
*p7 = &B7;
void F7() { int i1,i2,i3,i4;
i1 = A7.dd.mm; i2 = A7.pd->yy;
i3 = p7->dd.dd; i4 = p7->pd->yy; }
```

```
//------------------------------------------------------ 8
struct man9 {
char name[20];
char *zodiak;
man9 *next;
     } A9= {"Петров","Весы",NULL },
B9= {"Сидоров","Дева",&A9 },
*p9[4] = { &B9, &A9, &A9, &B9 };
void F9() { char c1,c2,c3,c4;
c1 = p9[0]->name[2]; c2 = p9[2]->zodiak[3];
c3 = p9[3]->next->name[3]; c4 = p9[0]->next->zodiak[1]; }
```



# 2. Иерархия типов и переменных

```
//-------------------------------------------------------54-01.cpp
// Иерархия типов данных и функций
// Уровень 1. Представление даты
struct date{ int dd,mm,yy;}
// Уровень 1. Представление строки таблицы (записи)
const int NP=100;
struct user{
        char name[20],pass[10];          // имя и пароль
        date birth;                      // дата регистрации
        int np;                          // текущая размерность в массива дат
        date SS[NP];                     // массив дат посещений
        double credit;};
// Уровень 3 - представление таблицы
struct table{
        user TBL[N];                     // масс
        int  nn;};                       // текущ
// Уровень 4 - основная программа
table TT;
```




любое выражение
имеет свой тип данных
**TT.TBL[i] - user**

# 2. Иерархия типов и переменных

```
//-------------------------------------------------------54-01.cpp
// Иерархия типов данных и функций
// Уровень 1. Представление даты
struct date{
    int dd,mm,yy;
    void setDate(int dd0, int mm0, int yy0){
        dd=dd0; mm=mm0; yy=yy0; }
    void getDate(){
        printf("\nday:"); scanf("%d",&dd);
        printf("month:"); scanf("%d",&mm);
        printf("year:"); scanf("%d",&yy);
        }
    int cmpDate(date &T){
        if (yy!=T.yy) return yy-T.yy;
        if (mm!=T.mm) return mm-T.mm;
        return dd-T.dd;
        }
    void loadDate(FILE *fd){ fscanf(fd,"%d%d%d",&dd,&mm,&yy); }
    void saveDate(FILE *fd){ fprintf(fd,"%d %d %d ",dd,mm,yy); }
    void showDate(){ printf("%02d.%02d.%04d ",dd,mm,yy); }
    int testDate(){
        if (dd<1 || dd>31 || mm<1 || mm>12) return 0;
        if (dd==30 && (mm==2 || mm==4 || mm==6 || mm==9 || mm==11)) return 0;
        if (dd==29 && mm==2 && yy%4==0) return 1;
        return 1;
        }
    };
```

# 2. Иерархия типов и переменных

```
// Уровень 1. Представление строки таблицы (записи)
const int N=100;
const int NP=100;
struct user{
    char name[20],pass[10];
    date birth;
    int np;
    date SS[NP];
    double credit;
    void setUser(char nm[], char ps[], date
        strcpy(name,nm);
        strcpy(pass,ps);
        birth=b0;
        np=0;
        }
    void addDate(date d0){
        if (np!=NP) SS[np++]=d0; }
    void showUser(){
        printf("%20s  %10s   ",name,pass);
        birth.showDate();
        printf("%2d\n",np);
        }
    void showDate(){
        for (int i=0;i<np;i++) {
            SS[i].showDate(); printf("\n"); }
        }
```

```
    void loadUser(FILE *fd){
        fscanf(fd,"%s%s",name,pass);
        birth.loadDate(fd);
        fscanf(fd,"%d",&np);
        for (int i=0;i<np;i++) SS[i].loadDate(fd);
        }
    void saveUser(FILE *fd){
        fprintf(fd,"%s\n%s\n",name,pass);
        birth.saveDate(fd);
        fprintf(fd,"\n%d\n",np);
        for (int i=0;i<np;i++) SS[i].saveDate(fd);
        fprintf(fd,"\n");
        }
    int cmpUser(user &T,int mode){
        switch (mode){
case 0: return strcmp(name,T.name);
case 1: return birth.cmpDate(T.birth);
case 2: return np-T.np;
        }}
    };
```

# 2. Иерархия типов и переменных

```
// Уровень 3 - представление таблицы
struct table{
    user TBL[N];
    int  nn;
    void addUser(char nm[], char ps[], date b0){
        if (nn!=NP) { TBL[nn].setUser(nm,ps,b0); nn++; }
        }
    void loadTable(char nm[]){
        FILE *fd=fopen(nm,"r");
        if (fd==NULL) return;
        fscanf(fd,"%d\n",&nn);
        if (nn>=N) return;
        for (int i=0;i<nn;i++) TBL[i].loadUser(fd);
        fclose(fd);
        }
    void saveTable(char nm[]){
        FILE *fd=fopen(nm,"w");
        if (fd==NULL) return;
        fprintf(fd,"%d\n",nn);
        for (int i=0;i<nn;i++) TBL[i].saveUser(fd);
        fclose(fd);
        }
    void sortTable(int mode){
        int i,j,k;
        for (i=0;i<nn;i++){
            for(j=k=i; j<nn;j++)
                if (TBL[j].cmpUser(TBL[k],mode)<0)
                    k=j;
            user cc=TBL[i]; TBL[i]=TBL[k]; TBL[k]=cc;
            }
    }};
```

# 2. Иерархия типов и переменных,

```
// Уровень 4 - основная программа
table TT;
void main(){
    int i;
    TT.nn=0;
    char c1[80],c2[80];
    date d0={1,1,2001};
    TT.addUser("admin","12345",d0);
    while(1){
        printf("\na(dd),v(iew),l(oad),s(ave),o(rd by),d(ate)\nwhat to do:");
        switch(getch()){
case 27:    return;              // escape - выход
case 'a':   printf("\nname:"); scanf("%s",c1);
            printf("password:"); scanf("%s",c2);
            d0.getDate();
            TT.addUser(c1,c2,d0);
            break;
case 'v':   printf("\nnn                name     password  birth       np\n");
            for (i=0;i<TT.nn;i++) {
            printf("%-2d",i);
            TT.TBL[i].showUser();}
            break;
case 's':   TT.saveTable("54-01.txt");
            break;
case 'l':   TT.loadTable("54-01.txt");
            break;
case 'o':   printf("\nsort mode (0-by name, 1-by date, 2-by count(np)):");
            TT.sortTable(getch()-'0');
            break;
```

# 3. Система типов данных в ЯП и в Си

## Определение типа переменной в контексте ее использования

- **контекст** – окружение, определяющее смысл фразы (выражение, вырванное из контекста)
- определение переменной содержит имя переменной в окружении (контексте) операций **выделения составляющего типа данных**, последовательно выполняемых над ней
  - операции записываются в той последовательности, в которой они могут быть применены к переменной, с учетом их естественных приоритетов и приоритетных скобок
  - в результате получается тип данных, стоящий слева в определении

| Тип данных | Операция | Использование в контекстном определении | Другие операции |
|---|---|---|---|
| Массив | **p[i]** – доступ по индексу | Да | |
| Указатель | ***p** - переход к указуемому объекту, разыменование | Да | **&a** - получение указателя (взятие адреса) |
| Структура | **p.name** (точка) – доступ по имени | Нет | **p->name** доступ по имени через указатель (сочетание * и точка) |
| Функция | **p(…)** – вызов функции | Да | |

# 3. Система типов данных в ЯП и в Си

Примеры контекстного определения

**int *p;** -указатель на целое

**char *p[];** - массив указателей на символы (строки)

**char (*p)[][80]; -** указатель на двумерный массив строк по 80 символов в строке

**int (*p)();** **-** указатель на функцию, возвращающую целое

**int (*p[10])();** -массив указателей на функции, возвращающих целое

**char *(*(*p)())());** - указатель на функцию, возвращающую в качестве результата указатель на функцию, возвращающую указатель на строку

**void (pp[10])(); - Массив функций – не бывает !!!!**

**int []F(){} - Функция, возвращающая массив – не бывает !!!!**

# 3. Система типов данных в ЯП и в Си

Где используется:

- определения и описания переменных – **int a[];**
- формальные параметры функций – void F(**int a[]**)
- результат функции;
- определения элементов структуры (**struct) –** struct user { **int a[10];** }
- определения абстрактных типов данных (АТД)
- определения промежуточных типов данных (спецификатор **typedef)**

**АТД** = контекстное определение переменное без имени переменной

- в операции **sizeof** – sizeof(**int***)
- в операторе создания динамических переменных **new** – new **int*[n]**
- в операции явного преобразования типа данных – ***(int*)0x1000**
- при объявлении формальных параметров внешней функции с использованием прототипа – extern int cmpstr(**char*, char***)

**typedef <определение переменной> -** вместо имени переменной определяет **имя типа данных,** обозначающего тип этой переменной (промежуточное имя, синоним)

```
typedef char *PSTR;      // PSTR - имя производного типа данных – указатель на char или char*
PSTR p, q[20],*pp;       // Эквивалентно char *p, *q[20], **pp;
```

# 3. Система типов данных в ЯП и в Си

Примеры

```
//-----------------------------------------------------55-0
//-----------------------------------------------------1
char f(void);
//-----------------------------------------------------2
char *f(void);
//-----------------------------------------------------3
int (*p[5])(void);
//-----------------------------------------------------4
void ( *(*p)(void) )(void);
//-----------------------------------------------------5
int (*f(void))();
//-----------------------------------------------------6
char **f(void);
//-----------------------------------------------------7
typedef char *PTR;
PTR a[20];
//-----------------------------------------------------8
typedef void (*PTR)(void);
PTR F(void);
//-----------------------------------------------------9
typedef void (*PTR)(void);
PTR F[20];
//-----------------------------------------------------10
struct list {...};
list *F(list *);
//-----------------------------------------------------11
void **p[20];
//-----------------------------------------------------12
char *(*pf)(char *);
//-----------------------------------------------------13
int F(char *,...);
//-----------------------------------------------------14
char **F(int);
//-----------------------------------------------------15
typedef char *PTR;
PTR F(int);
```