

# Лекция №4

*Еще кое что про  
символы и строки*

Преобразование символов (ASCII)

Особенности ввода строк

*Структуры данных*

Создание пользовательских типов  
данных

*Разработка программ*

Метод «сверху-вниз»

Литература

# Символьный тип данных

В вычислительных машинах символы не могут храниться иначе, как в виде последовательностей бит (как и числа).



**char** – тип данных

В языке Си размер типа равен **одному байту**

```
void main()
```

```
{  
    char a;  
    for(a = 65; a < 91; a++)  
        printf("%d\t", a);  
}
```

65	66	67	68	69	70	71	72	73	74
75	76	77	78	79	80	81	82	83	84
85	86	87	88	89	90				

Для продолжения нажмите любую клавишу . . .

```
void main()
```

```
{  
    char a;  
    for(a = 65; a < 91; a++)  
        printf("%c\t", a);  
}
```

A	B	C	D	E	F	G	H	I	J
K	L	M	N	O	P	Q	R	S	T
U	V	W	X	Y	Z				

Для продолжения нажмите любую клавишу . . .

# Таблица кодировки ASCII

**ASCII** (*American standard code for information interchange*)

название таблицы (кодировки, набора), в которой некоторым распространённым печатным и непечатным символам сопоставлены числовые коды.

Таблица была стандартизована в 1963 г.

Таблица ASCII определяет коды для:

- десятичных цифр;
- латинского алфавита;
- национального алфавита;
- знаков препинания;
- управляющих символов.

0 -	16 - ►	32 -	48 - 0	64 - @	80 - P	96 - '	112 - p
1 - ☺	17 - ◀	33 - !	49 - 1	65 - A	81 - Q	97 - a	113 - q
2 - ☼	18 - ⇅	34 - "	50 - 2	66 - B	82 - R	98 - b	114 - r
3 - ♥	19 - !!	35 - #	51 - 3	67 - C	83 - S	99 - c	115 - s
4 - ♦	20 - ¶	36 - \$	52 - 4	68 - D	84 - T	100 - d	116 - t
5 - ♣	21 - ♂	37 - %	53 - 5	69 - E	85 - U	101 - e	117 - u
6 - ♠	22 - ▬	38 - &	54 - 6	70 - F	86 - V	102 - f	118 - v
7 -	23 - ⇅	39 - '	55 - 7	71 - G	87 - W	103 - g	119 - w
8 -	24 - ↑	40 - (	56 - 8	72 - H	88 - X	104 - h	120 - x
9 -	25 - ↓	41 - )	57 - 9	73 - I	89 - Y	105 - i	121 - y
10 -	26 - →	42 - *	58 - :	74 - J	90 - Z	106 - j	122 - z
11 -	27 - ⇐	43 - +	59 - ;	75 - K	91 - [	107 - k	123 - {
12 -	28 - ⇐	44 - ,	60 - <	76 - L	92 - \	108 - l	124 -
13 -	29 - ⇔	45 - -	61 - =	77 - M	93 - j	109 - m	125 - }
14 - ♪	30 - ▲	46 - .	62 - >	78 - N	94 - ^	110 - n	126 - ~
15 - ☼	31 - ▼	47 - /	63 - ?	79 - O	95 - ÷	111 - o	127 - ␣
16 - ►	32 -	48 - 0	64 - @	80 - P	96 -	112 - p	

128 - A	144 - P	160 - a	176 - ☒	192 - L	208 - ⌚	224 - p	240 - Ė
129 - Б	145 - C	161 - б	177 - ☒	193 - ⌚	209 - ⌚	225 - c	241 - ė
130 - В	146 - Т	162 - в	178 - ▒	194 - ⌚	210 - ⌚	226 - т	242 - ė
131 - Г	147 - У	163 - г	179 -	195 - ⌚	211 - ⌚	227 - у	243 - ė
132 - Д	148 - Ф	164 - д	180 - }	196 - —	212 - ⌚	228 - ф	244 - Ė
133 - Е	149 - Х	165 - е	181 - }	197 - ⌚	213 - ⌚	229 - х	245 - Ė
134 - Ж	150 - Ц	166 - ж	182 - ⌚	198 - ⌚	214 - ⌚	230 - ц	246 - Ÿ
135 - З	151 - Ч	167 - з	183 - ⌚	199 - ⌚	215 - ⌚	231 - ч	247 - Ÿ
136 - И	152 - Ш	168 - и	184 - ⌚	200 - ⌚	216 - ⌚	232 - ш	248 - Ÿ
137 - Й	153 - Щ	169 - й	185 - ⌚	201 - ⌚	217 - ⌚	233 - щ	249 - ●
138 - К	154 - Ъ	170 - к	186 - ⌚	202 - ⌚	218 - ⌚	234 - ъ	250 - .
139 - Л	155 - Ы	171 - л	187 - ⌚	203 - ⌚	219 - ⌚	235 - ы	251 - √
140 - М	156 - Ь	172 - м	188 - ⌚	204 - ⌚	220 - ⌚	236 - ь	252 - №
141 - Н	157 - Э	173 - н	189 - ⌚	205 - ⌚	221 - ⌚	237 - э	253 - ¤
142 - О	158 - Ю	174 - о	190 - ⌚	206 - ⌚	222 - ⌚	238 - ю	254 - ■
143 - П	159 - Я	175 - п	191 - ⌚	207 - ⌚	223 - ⌚	239 - я	255 -
144 - Р	160 - а	176 - ☒	192 - L	208 - ⌚	224 - p	240 - Ė	



# Таблица кодировки ASCII [0-127]

- Последовательное размещение цифр [48 – 57]
- Последовательное размещение заглавных букв [65 – 90]
- Последовательное размещение прописных букв [97 – 122]
- $((\text{int})'a' - (\text{int})'A') = 32$

0 -	16 - ►	32 -	48 - 0	64 - @	80 - P	96 - '	112 - p
1 - ☹	17 - ◀	33 - !	49 - 1	65 - A	81 - Q	97 - a	113 - q
2 - ☹	18 - ⬆	34 - "	50 - 2	66 - B	82 - R	98 - b	114 - r
3 - ♥	19 - !!	35 - #	51 - 3	67 - C	83 - S	99 - c	115 - s
4 - ♦	20 - ¶	36 - \$	52 - 4	68 - D	84 - T	100 - d	116 - t
5 - ♣	21 - Ⓞ	37 - %	53 - 5	69 - E	85 - U	101 - e	117 - u
6 - ♣	22 - ▬	38 - &	54 - 6	70 - F	86 - V	102 - f	118 - v
7 -	23 - ⬆	39 - '	55 - 7	71 - G	87 - W	103 - g	119 - w
8 -	24 - ⬆	40 - (	56 - 8	72 - H	88 - X	104 - h	120 - x
9 -	25 - ⬇	41 - )	57 - 9	73 - I	89 - Y	105 - i	121 - y
10 -	26 - ➡	42 - *	58 - :	74 - J	90 - Z	106 - j	122 - z
11 -	27 - ◀	43 - +	59 - ;	75 - K	91 - [	107 - k	123 - {
12 -	28 - ▬	44 - ,	60 - <	76 - L	92 - \	108 - l	124 -
13 -	29 - ➡	45 - -	61 - =	77 - M	93 - j	109 - m	125 - }
14 - 🎵	30 - ▲	46 - .	62 - >	78 - N	94 - ^	110 - n	126 - ~
15 - ☼	31 - ▼	47 - /	63 - ?	79 - O	95 - ÷	111 - o	127 - ␣
16 - ►	32 -	48 - 0	64 - @	80 - P	96 -	112 - p	

```
void main()
{
    char ch;
    scanf("%c", &ch);
    if(ch >= 97 && ch <= 122)
        ch = ch - 32;
    printf("%c", ch);
}
```

```
void main()
{
    char ch;
    scanf("%c", &ch);
    if(ch >= 'a' && ch <= 'z')
        ch -= ('a' - 'A');
    printf("%c", ch);
}
```

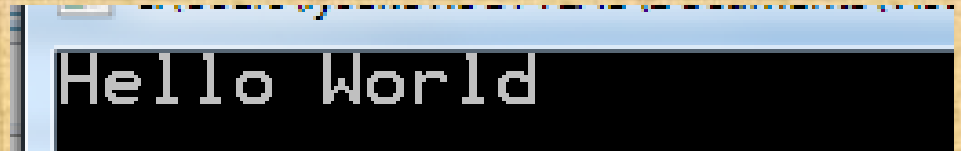
# Строки в Си

```
printf("Hello World");
```

0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o		w	o	r	l	d		

*Строка в Си – одномерный массив символов(\*)*

```
void main()  
{  
    int i;  
    char str[] = "Hello World";  
  
    for(i = 0; i < 11; i++)  
        printf("%c", str[i]);  
}
```



# Строки в Си

```
void main()
```

```
{
```

```
    char str[] = "Hello World";
```

```
    printf("%d", sizeof(str));
```

```
}
```

0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o			w	o	r	l	d	

12

**нуль-символ '\0'**

при объявлении строковой константы  
нуль-символ добавляется автоматически

0	1	2	3	4	5	6	7	8	9	10	11	12
H	e	l	l	o			w	o	r	l	d	\0

```
void main()
```

```
{
```

```
    char str[] = "Hello World";
```

```
    printf("%s", str);
```

```
}
```

Автоматический контроль  
окончания строки:  
вывод происходит до символа '\0'

# Ввод и вывод строк в Си

*(\*) Строка в языке Си – одномерный массив символов, заканчивающийся '\0'*

*%s – спецификатор работы со строками при вводе и выводе в языке Си*

```
void main()
{
    char str[80];
    scanf("%s", str);
    printf("%s", str);
}
```

***Контроль размера массива  
количество элементов всегда должно быть больше  
количества хранящихся символов***



# Присвоение значений строкам в Си

```
char str[] = "Hello World";
```

- Автоматическое вычисление размера массива
- Автоматическое добавление '\0'

```
char str[12] = "Hello World";
```

- Автоматическое добавление '\0' (\*)
- **отсутствие контроля корректности размера**

```
void main()  
{  
    char str[10] = "Hello World";  
    printf("%s", str);  
}
```



```
void main()  
{  
    char str[3];  
    str[0] = 'H';  
    str[1] = 'i';  
    str[2] = '\0';  
    printf("%s", str);  
}
```

- **Ручной контроль**



# Работа со строкам в Си

- *Подсчет количества символов в строке*

```
int Count(char* pStr)
{
    int i;
    for(i = 0; *(pStr+i)!='\0'; i++);
    return i;
}
```

```
void SetAllBig(char* pStr)
{
    for(int i = 0; *(pStr+i) != '\0'; i++)
        if(*(pStr+i) >= 'a' && *(pStr+i) <= 'z')
            *(pStr+i) -= 32;
}
```

- *Замена всех прописных букв на заглавные*

# Работа со строкам в Си

```
char* GetString()  
{
```

```
    char* pStr;  
    char temp[20];  
    int i, size;  
    scanf("%s", temp);
```

```
    for(size = 0; *(temp+size) != '\0'; size++);
```

```
    pStr = malloc((size+1)*sizeof(char));
```

```
    for(i = 0; *(temp+i) != '\0'; i++)  
        *(pStr+i)=temp[i];  
    *(pStr+size) = '\0';
```

```
    return pStr;
```

```
}
```

- *Динамические строки*

- **Ввод только одного слова**  
( до пустого символа,  
которым может быть  
пробел, табуляция или  
перевод строки)

- **Ограничение на  
максимальный размер**

- **Отсутствие контроля за  
превышением размера**

```
scanf("%19s", temp);
```

- *Разбиение ввода на части*

# Работа со строкам в Си

```
char* GetString()
{
    char* pStr;
    char temp[20];
    int i, size;
    gets(temp);

    for(size = 0; *(temp+size) != '\0'; size++);

    pStr = malloc((size+1)*sizeof(char));

    for(i = 0; *(temp+i) != '\0'; i++)
        *(pStr+i)=temp[i];
    *(pStr+size) = '\0';

    return pStr;
}
```

- *Динамические строки*
  - **Ввод текста**  
(до перевода строки (ENTER))
  - **Отсутствие контроля за превышением размера**

**Возможно  
переполнение массива и  
несанкционированная  
запись в память**

# Работа со строкам в Си

```
char* GetString()
{
    char* pStr;
    char temp[20];
    int i, size;
    fgets(temp, 20, stdin);

    for(size = 0; *(temp+size) != '\0'; size++);

    pStr = malloc((size+1)*sizeof(char));

    for(i = 0; *(temp+i) != '\0'; i++)
        *(pStr+i)=temp[i];
    *(pStr+size) = '\0';

    return pStr;
}
```

- *Динамические строки*
  - **Ввод текста**  
(до перевода строки (ENTER))
  - **Корректная обработка превышения ввода**  
(разбиение на части)



# Работа со строкам в Си

```
char* GetString()
{
    char* pStr;
    char temp[20];
    char ch;
    int i, size;
    for(i = 0; (int)ch != 10; i++)
    {
        ch = getchar();
        temp[i] = ch;
    }
    temp[i] = '\0';

    for(size = 0; *(temp+size) != '\0'; size++);
    pStr = malloc((size+1)*sizeof(char));
    for(i = 0; *(temp+i) != '\0'; i++)
        *(pStr+i)=temp[i];
    *(pStr+size) = '\0';
    return pStr;
}
```

- *Динамические строки*
  - **Посимвольный ввод**
  - **10 – код «перевода строки»**
  - **Возможность ручного контроля превышения размера буфера**

# Работа со строкам в Си

```
char* pStr;  
pStr = GetString();
```

- *Вывод строк на консоль*

```
printf("%s", p);
```

```
puts(p);
```

```
fputs(p, stdin);
```

```
for(int i = 0; *(p+i)!='\0'; i++)  
    putchar(*(p+i));
```

???

# Многомерные массивы данных

Одномерные массивы

*int Mas[10];*

Двумерные массивы

```
int array2D[3][3]={10,20,30,  
                    40,50,60,  
                    70,80,90};
```

[0]	10	20	30
[1]	40	50	60
[2]	70	80	90
	[0]	[1]	[2]

Многомерные массивы

*int Mas[Z][Y][X];*

# Двумерные массивы данных

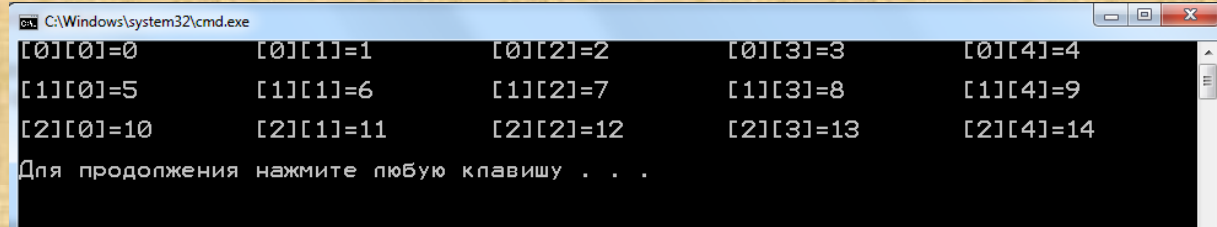
```
#include <stdio.h>  
#define Width 5  
#define Height 3  
void main()  
{
```

```
    int myArray[Height][Width];  
    int x, y;
```

```
    for ( y = 0; y < Height; y++ )  
        for ( x = 0; x < Width; x++ )  
            myArray[y][x] = y*Width + x;
```

```
    for ( y = 0; y < Height; y++ )  
    {  
        for ( x = 0; x < Width; x++ )  
            printf( "[%d][%d]=%d\t", y, x, myArray[y][x] );  
            printf( "\n" );  
    }
```

```
}
```



```
C:\Windows\system32\cmd.exe  
[0][0]=0      [0][1]=1      [0][2]=2      [0][3]=3      [0][4]=4  
[1][0]=5      [1][1]=6      [1][2]=7      [1][3]=8      [1][4]=9  
[2][0]=10     [2][1]=11     [2][2]=12     [2][3]=13     [2][4]=14  
Для продолжения нажмите любую клавишу . . .
```



# Двумерные массивы данных

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

```
void PrintMassive(int* pMas, int Height, int Width)
```

```
{
```

```
    int x, y;
```

```
    for (y = 0; y < Height; y++ )
```

```
    {
```

```
        for (x = 0; x < Width; x++ )
```

```
            printf( "%d\t", *(pMas+y*Width+x));
```

```
        printf( "\n" );
```

```
    }
```

```
}
```

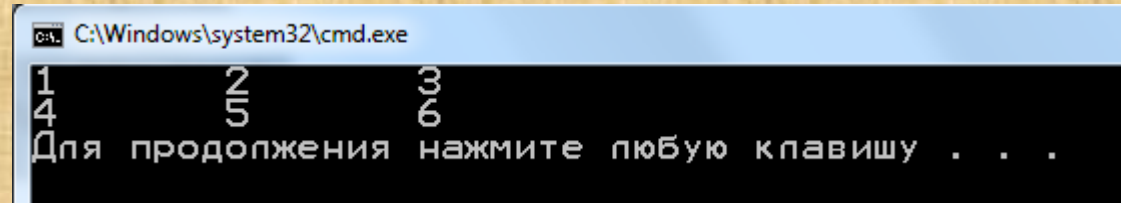
```
int main()
```

```
{
```

```
    int myArray[2][3] = {{1,2,3},{4,5,6}};
```

```
    PrintMassive(myArray, 2, 3);
```

```
}
```



```
C:\Windows\system32\cmd.exe
1      2      3
4      5      6
Для продолжения нажмите любую клавишу . . .
```

# Двумерные массивы данных

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

```
void PrintMassive(int* pMas, int Height, int Width)
```

```
{
```

```
    int x, y;
```

```
    for (y = 0; y < Height; y++)
```

```
    {
```

```
        for (x = 0; x < Width; x++)
```

```
            printf( "%d\t", *(pMas+y*Width+x));
```

```
        printf( "\n" );
```

```
    }
```

```
}
```

```
int main()
```

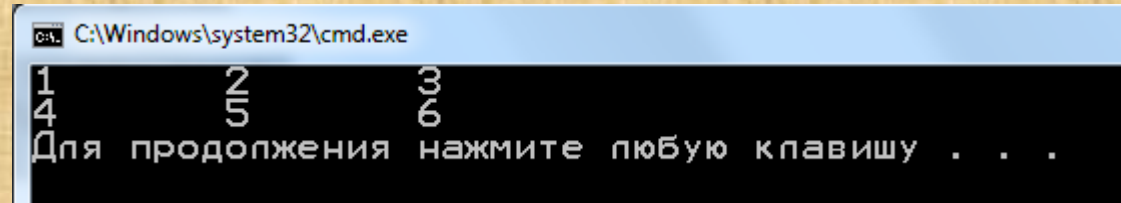
```
{
```

```
    int myArray[2][3] = {{1,2,3},{4,5,6}};
```

```
    PrintMassive(myArray, 2, 3);
```

```
}
```

???



```
C:\Windows\system32\cmd.exe
1      2      3
4      5      6
Для продолжения нажмите любую клавишу . . .
```

```
int myArray[6] = {1,2,3,4,5,6};
```

# Группы переменных

Однотипные переменные

массивы

```
int mas[24];
```

```
char* pString
```

Произвольные типы

структуры

```
struct Book  
{
```

```
int    pages;  
char author[50];  
char title[100];  
float price;
```

```
};
```

Структурный тип данных -

один из способов создания своих типов данных в языке Си

# Структуры данных в языке Си

**Структура** - это совокупность переменных, объединенных одним именем, предоставляющая общепринятый способ совместного хранения информации.

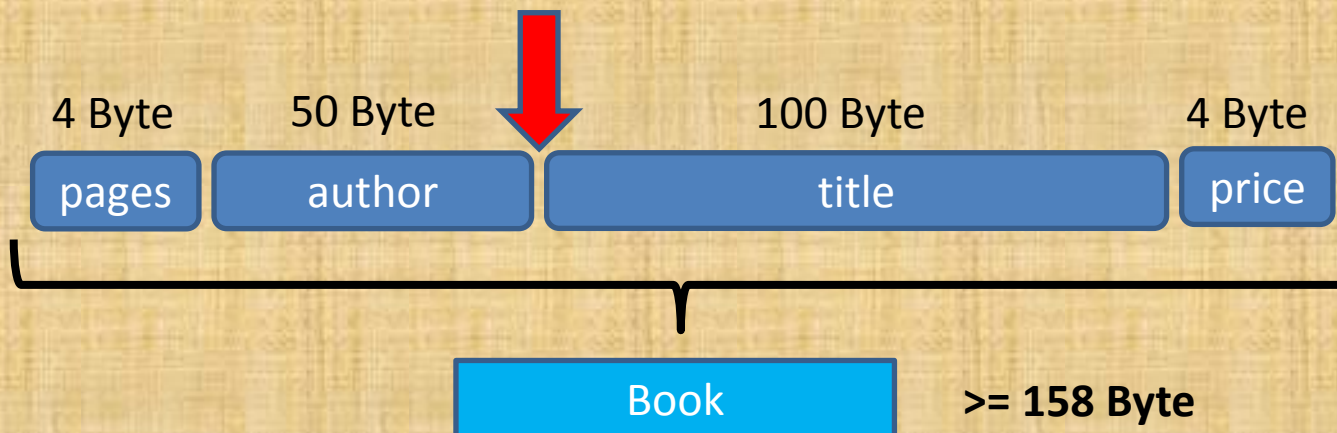
Объявление структуры приводит к образованию **шаблона**, используемого для создания **объектов структуры**.

Переменные, образующие структуру, называются членами структуры или **полями структуры**.

```
struct Book  
{
```

```
    int    pages;  
    char  author[50];  
    char  title[100];  
    float price;
```

```
};
```





# Структуры данных в языке Си

```
struct Book
```

```
{
```

```
    int  pages;
```

```
    char author[50];
```

```
    char title[100];
```

```
    float price;
```

```
};
```

Структурный тип данных Book

pages - поле структуры типа int

```
void main()
```

```
{
```

```
    struct Book A;
```

```
    int size;
```

A – объект

```
    A.pages = 78;
```

```
    scanf("%s", A.author);
```

```
    scanf("%s", A.title);
```

```
    scanf("%f", &A.price);
```

Доступ к полям объекта структуры  
через (.)

```
    size = sizeof(struct Book);
```

```
}
```

# Структуры данных в языке Си

```
struct Book
```

```
{
```

```
    int    pages;
```

```
    char  author[50];
```

```
    char  title[100];
```

```
    float price;
```

```
};
```

Структурный тип данных Book

pages - поле структуры типа int

```
void main()
```

```
{
```

```
    struct Book A;
```

```
    struct Book* pA;
```

A – объект

pA – указатель на объект структуры

```
    pA = &A;
```

Доступ к полям структуры через  
указатель на объект (->)

```
    pA->pages = 78;
```

```
    scanf("%s", pA->author);
```

```
    scanf("%s", pA->title);
```

```
    scanf("%f", &(pA->price));
```

```
}
```

# Структуры данных в языке Си

```
struct Book
```

```
{
```

```
    int    pages;
```

```
    char  author[50];
```

```
    char  title[100];
```

```
    float price;
```

```
} Book1;
```

```
void main()
```

```
{
```

```
    struct Book* pA;
```

```
    pA = &Book1;
```

```
    pA->pages = 78;
```

```
    scanf("%s", pA->author);
```

```
    scanf("%s", pA->title);
```

```
    scanf("%f", &(pA->price));
```

```
}
```

**Структурный тип данных Book**

**Глобальная переменная Book1 –  
объект структуры**

**pA – указатель на объект структуры**

**Доступ к полям структуры через  
указатель на объект (->)**

# TYPEDEF - определение имени новых типов данных

```
typedef struct  
{
```

**Book – имя нового типа данных**

```
    int pages;  
    char author[50];  
    char title[100];  
    float price;
```

```
} Book;
```

**Возможность инициализации  
при создании объекта**

```
void main()  
{
```

```
    Book A = {78, "Brian W. Kernighan, Dennis M. Ritchie",  
             "The C programming Language", 15.2};
```

```
    printf("%s\t%s\t%d\t%f\n", A.author, A.title, A.pages, A.price);
```

```
}
```



# Работа со структурами

```
typedef struct  
{  
    int pages;  
    char author[50];  
} Book;
```

```
void InitBookMas(Book* pMas, int size)  
{  
    for(int i = 0; i < size; i++)  
        scanf("%s%d", (pMas+i)->author, &((pMas+i)->pages));  
}
```

```
void main()  
{  
    Book mas[10];  
    InitBookMas(mas, 7);  
    mas[7].pages = 78;  
}
```

Как с обычными типами данных:

- объявление
- передача в качестве параметров
- возвращение значений



# Структурное программирование

*В 1968 году Эдсгер Виле Дейкстра  
опубликовал свое знаменитое письмо  
«Оператор GoTo считается вредным»*

методология разработки программного обеспечения,  
в основе которой лежит представление программы  
в виде иерархической структуры блоков.  
Предложена в 1970-х годах Э. Дейкстрой и др.

- Любая программа строится без использования оператора goto
- Три базовых управляющих структуры: последовательность, ветвление, цикл
- Использование подпрограмм (функций)
- Разработка программы ведётся пошагово, методом «сверху вниз»

# Метод «сверху вниз»

1. Сначала пишется текст основной программы, в котором, вместо каждого связного логического фрагмента текста, вставляется вызов подпрограммы.
2. Вместо настоящих, работающих подпрограмм, в программу вставляются фиктивные части — заглушки.
  - заглушка удовлетворяет требованиям интерфейса заменяемого фрагмента (модуля), но не выполняет его функций или выполняет их частично.
3. После того, как программист убедится, что подпрограммы вызываются в правильной последовательности (то есть общая структура программы верна), подпрограммы-заглушки последовательно заменяются на реально работающие.
  - разработка каждой подпрограммы ведётся тем же методом, что и основной программы.

Такая последовательность гарантирует, что на каждом этапе разработки программист одновременно имеет дело с обозримым и понятным ему множеством фрагментов.

При сопровождении и внесении изменений в программу выясняется, в какие именно процедуры нужно внести изменения. Они вносятся, не затрагивая части программы, непосредственно не связанные с ними. Это гарантирует, что при внесении изменений и исправлении ошибок не выйдет из строя часть программы, находящаяся в данный момент вне зоны внимания.



# Постановка задачи

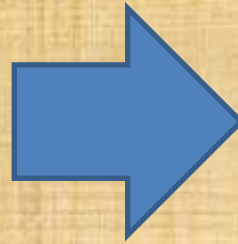
**0! Выяснить особенности поставленной задачи – сформулировать ТЗ**

**Задача:**

**Отсортировать введенной пользователем  
массив и распечатать на экран**

Уточнения:

1. Какие данные, диапазон
2. Размер массива – кто задает
3. Способ сортировки
  - Трудоемкость
  - Где хранить данные
  - Предварительные знания



- Целочисленные данные
- Диапазон от -1000 до 5000
- Размер задает пользователь
- Скорость сортировки не важна
- Результат сортировки в отдельном массиве

- *Тип данных int*
- *Хранение в виде динамического массива*
- *Размер задается с консоли*
- *Данные вводятся с консоли, необходим контроль диапазона*
- *Создание второго массива с результатами*
- *Сортировка пузырьком*
- *Вывод массива на консоль*



# 1. Текст основной программы, в котором, вместо каждого связного логического фрагмента текста, вставляется вызов подпрограммы.

```
void main()
{
    int *pMas, *pResMas;
    int size, order;

    printf("Input size: ");
    scanf("%d", &size);
    pMas = AllocateMemory(size);

    InitAndCheckMassive(pMas, size, -1000, 5000);

    pResMas = AllocateMemory(size);
    printf("0 to ascending order, else 1");
    scanf("%d", &order);
    BubbleSort(pMas, pResMas, size, order);

    PrintMas(pResMas, size);

    free(pMas);
    free(pResMas);
}
```

- Тип данных *int*
- Хранение в виде динамического массива
- Размер задается с консоли
- Данные вводятся с консоли, контроль диапазона
- Создание второго массива с результатами
- Сортировка пузырьком
- Вывод массива на консоль

- Имена переменных
- Имена функций
- Читаемость кода
- Понятность действий
- Параметры и возвращаемые значения
- Контроль памяти

## 2. Добавление «заглушек» функций

```
int* AllocateMemory(size)
{
    int* p = NULL;
    return p;
}

void InitAndCheckMassive(int* pMas, int size, int Min, int Max)
{
}

void BubbleSort(int* pSrcMas, int* pResMas, int size, int order)
{
}

void PrintMas(int* pMas, int size)
{
}

void main()
{
    .....
    pMas = AllocateMemory(size);
    InitAndCheckMassive(pMas, size, -1000, 5000);
    .....
    pResMas = AllocateMemory(size);
    BubbleSort(pMas, pResMas, size, order);
    .....
    PrintMas(pResMas, size);
    .....
}
```

*заглушка* удовлетворяет требованиям интерфейса заменяемого фрагмента (модуля), но не выполняет его функций или выполняет их частично

### Цель:

1. Компиляция
  2. Сборка (линковка)
  3. Запуск программы
  4. Исполнение программы
- **Корректное завершение**

### 3. Заглушки дополняются необходимой функциональностью

```
int* AllocateMemory(int size)
```

```
{  
    int* p = NULL;  
    p = malloc(size * sizeof(int));  
    return p;  
}
```

```
void InitAndCheckMassive(int* pMas, int size, int Min, int Max)
```

```
{  
    int i, flag;  
    for(i = 0; i < size; i++)  
    {  
        do  
        {  
            printf("mas[%d] = ", i);  
            scanf("%d", (pMas+i));  
            if(*(pMas+i) >= Min && *(pMas+i) <= Max)  
                flag = 1;  
            else  
                flag = 0;  
        } while(!flag);  
    }  
}
```

- Имена переменных
  - Имена функций
  - Читаемость кода
  - Понятность действий
- 
- Объем кода для одной функции не должен превышать видимой области

### 3. Заглушки дополняются необходимой функциональностью

```
void CopyData(int* pSrcMas, int* pDstMas, int size)
{
}

int Compare(int* pMas, int i, int j)          // 1  [i] > [j]
{                                              // -1 [i] < [j]
    int res = 0;                             // 0  [i] = [j]
    return res;
}

void Replace(int* pMas, int i, int j)
{
}

void BubbleSort(int* pSrcMas, int* pResMas, int size, int order)
{
    int res, n, i;
    CopyData(pSrcMas, pResMas, size);
    for(n = 0; n < size-1; n++)
        for(i = 0; i < size-2; i++)
        {
            res = Compare(pResMas, i, i+1);
            if(res == order)
                Replace(pResMas, i, i+1);
        }
}
```

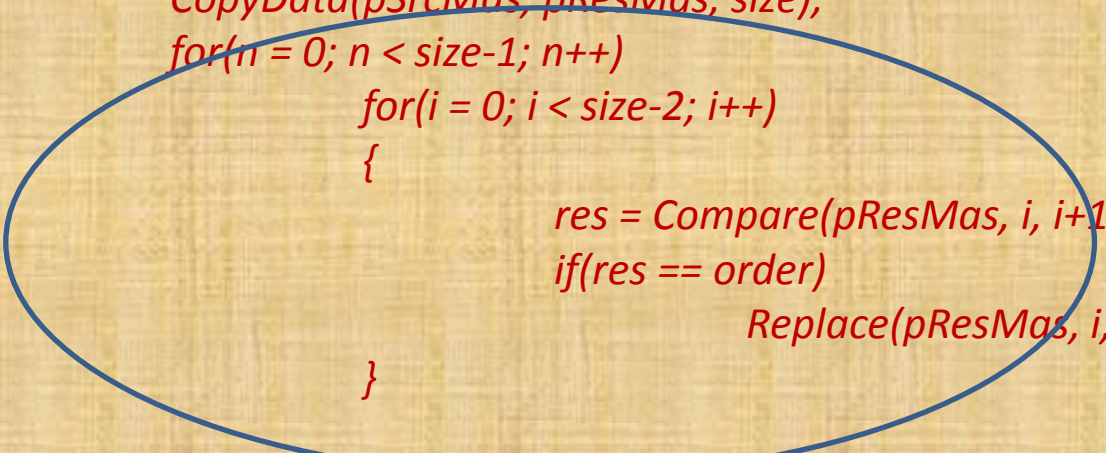
разработка каждой  
подпрограммы ведётся  
тем же методом, что  
и основной программы

**комментарии**  
**=**  
**документация**



**При сопровождении и внесении изменений в программу выясняется, в какие именно процедуры нужно внести изменения. Они вносятся, не затрагивая части программы, непосредственно не связанные с ними.**

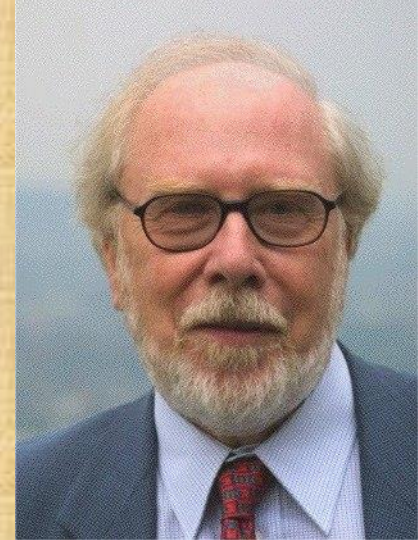
```
void BubbleSort(int* pSrcMas, int* pResMas, int size, int order)
{
    int res, n, i;
    CopyData(pSrcMas, pResMas, size);
    for(n = 0; n < size-1; n++)
        for(i = 0; i < size-2; i++)
        {
            res = Compare(pResMas, i, i+1);
            if(res == order)
                Replace(pResMas, i, i+1);
        }
}
```



**Изменения в алгоритме сортировки  
(оптимизации)  
затрагивают только одну функцию**

## Никлаус Вирт (Niklaus Wirth, род. 15 февраля 1934 года)

- швейцарский учёный, специалист в области информатики
- один из известнейших теоретиков в области разработки языков программирования
- создатель и ведущий проектировщик языков программирования Паскаль, Модула-2, Оберон...

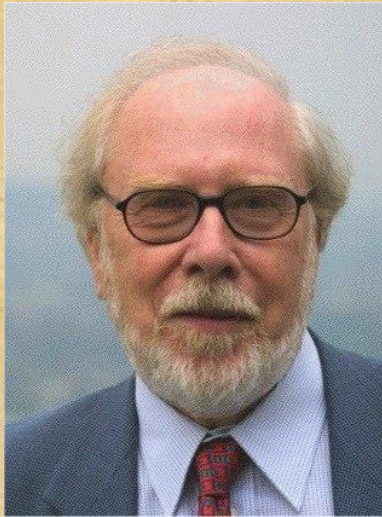


*«Инструмент должен соответствовать задаче. Если инструмент не соответствует задаче, нужно придумать новый, который бы ей соответствовал, а не пытаться приспособить уже имеющийся»*

***технология структурного программирования  
разработана, обоснована и внедрена в жизнь  
Виртом, Дейкстрой и Хоаром.***



# Литература



**Никлаус Вирт.**

## **Алгоритмы и структуры данных.**

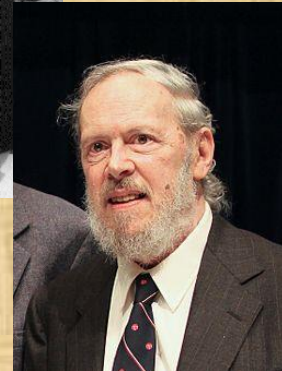
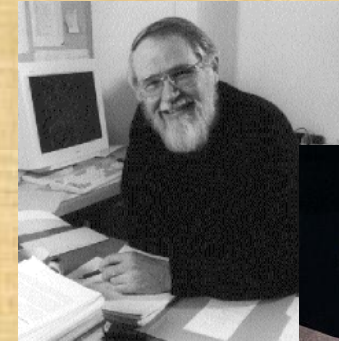
- фундаментальные принципы построения эффективных и надежных программ
- На примерах прорабатываются традиционные темы алгоритмики: сортировка и поиск, рекурсия, динамические структуры данных
- программы на языке Модула-2



**Дональд Эрвин Кнут.**

## **Искусство программирования.**

- фундаментальная монография
- в 1999 году признана одной из двенадцати лучших физико-математических монографий столетия
- том 1. Основные алгоритмы.
- том 2. Получисленные алгоритмы.
- том 3. Сортировка и поиск.
- том 4, А. Комбинаторные алгоритмы.....



**Брайан У. Керниган,**

**Деннис М. Ритчи.**

## **Язык программирования С.**

- Классическая книга по языку С, написанная самими разработчиками этого языка
- является как практически исчерпывающим справочником, так и учебным пособием
- Книга не рекомендуется для чтения новичкам (требует знания основ программирования и вычислительной техники)