# Програмен език C

Пламен Танов
Ненко Табаков
Мартин Вачовски
Технологично училище „Електронни системи“
Технически университет – София

версия 0.5

# *Литература*
# *Необходими програми*

Kernighan & Ritchie - The C Programming Language

MinGW
http://www.mingw.org/

Notepad++
http://notepad-plus.sourceforge.net/

# Кратка история

- Създаден през 1978 година
- Първоначално е създаден и имплементиран за UNIX операционната система
- През 1983 American National Standards Institute (ANSI) създава комитет с цел създаване на еднозначна дефиниция на езика C
- През 1988 излиза ANSI стандартът за C, наричан още “ANSI C”

# Въведение

Характеристики на езика C:

- език с общо предназначение
- типове за цели числа, числа с плаваща запетая и символи
- конструкции за управление на последователността
- възможност за създаване на собствени типове данни
- предпроцесорна обработка
- адресна аритметика
- лесно преносим (няма вградени в езика средства за вход и изход - те се осъществяват от библиотечни функции)
- максимално близък до системната архитектура (тип int, преобразуване на типовете, преместване в дясно, ...)

# Първа програма

```
#include <stdio.h>

int main () {
   printf ("Hello world\n");
   return 0;
}
```

# Пример

1°C = (5/9) * (1°F – 32)

```
#include <stdio.h>
/*
     print Fahrenheit-Celsius table
     for fahr = 0, 20, ..., 300
*/
int main() {
     int fahr, celsius;
     int lower, upper, step;
     lower = 0;        /* lower limit of temperature scale */
     upper = 300;      /* upper limit */
     step = 20;        /* step size */
     fahr = lower;
     while (fahr <= upper) {
         celsius = 5 * (fahr-32) / 9;
         printf("%d\t%d\n", fahr, celsius);
         fahr = fahr + step;
     }
     return 0;
}
```

# Изход на програмата

```
0 -17
20 -6
40 4
60 15
80 26
100 37
120 48
140 60
160 71
180 82
200 93
220 104
240 115
260 126
280 137
300 1481
```

# Коментари

Ограждат се в /* ... */ или с // и коментарът е до края на реда

```
/*
    това е коментар
*/

// и това е коментар, до края на реда
```

# Основни типове

- **`char`** – символен
- **`int`** – целочислен
- **`float`** – число с плаваща запетая и единична точност
- **`double`** – число с плаваща запетая и двойна точност

```
int fahr, other_int;
float celsius;
```

# Цикъл while

```
while (условие) {
    /*
          тяло, изпълнява се докато условието
  е истина (т.е. е различно от 0!)
    */
}
```

```
while (fahr <= upper) {
    celsius = 5 * (fahr - 32) / 9;
    printf ("%d\t%d\n", fahr, celsius);
    fahr = fahr + step;
}
```

# Оператор if

```
if (условие)
    оператор1;
else
    оператор2;
```

```
if (c >= '0' && c <= '9') {
    printf("digit");
} else {
    printf("not a digit");
}
```

# Функцията printf

```
printf ("символен низ", арг1, арг2, ...);
```

"символен низ":
%d – изписва целочислена стойност
%f – изписва реална стойност
%c – изписва символ
%s – изписва символен низ

\n – нов ред
\t – табулатор
\\ – \

# Пример

```
#include <stdio.h>

int main () {
    int fahr;

    for (fahr = 0; fahr <= 300; fahr = fahr + 20)
       printf("roll: %3d\t%6.1f\n", fahr,
                                (5.0/9.0)*(fahr-32));
    return 0;
}
```

# Изход на програмата

```
roll:   0        -17.8
roll:  20         -6.7
roll:  40          4.4
roll:  60         15.6
roll:  80         26.7
roll: 100         37.8
roll: 120         48.9
roll: 140         60.0
roll: 160         71.1
roll: 180         82.2
roll: 200         93.3
roll: 220        104.4
roll: 240        115.6
roll: 260        126.7
roll: 280        137.8
roll: 300        148.9
```

# Цикъл for

```
for (инициализация;условие;стъпка) {
    /* тяло, изпълнява се докато условието
  е истина (т.е. е различно от 0!) */
}
```

```
int i;
for (i = 0; i<=10; i = i + 2)
    printf ("%d\n", i);

i = 0;
for (; i<=10; i = i + 2)
    printf ("%d\n", i);

for (i = 0;i<=10;) {
    printf ("%d\n", i);
    i = i + 2;
}
```

# Директива #define

**#define ИМЕ СТОЈНОСТ**

```
#include <stdio.h>

#define LOWER 0
#define UPPER 300
#define STEP 20

int main () {
     int fahr;

     for (fahr = LOWER; fahr<=UPPER; fahr = fahr + STEP)
        printf ("%3d %6.1f\n", fahr, (5.0/9.0)*(fahr-32));
     return 0;
}
```

# *Изрази и приоритет на действие*

```
c = getchar()!= EOF;
```

еквивалентно на

```
c = (getchar()!= EOF);
```

**!=** – логически оператор „различно от“

**==** – логически оператор „еднакво с“

**=** – оператор за присвояване на стойност

и други

В Паскал съответно: **<>, =, :=**

# Примери

```
#include <stdio.h>
/* версия 1*/
int main () {
 int c;
 c = getchar ();
 while (c != EOF) {
  putchar(c);
  c = getchar ();
 }
 return 0;
}
```

```
#include <stdio.h>
/* версия 2*/
int main () {
 int c;

 while ((c = getchar ())!= EOF) {
  putchar(c);
 }

 return 0;
}
```

# Пренасочване на стандартния вход и изход

Това е функция на операционната система.
При Linux, Windows и DOS:


# Оператори за увеличаване и намаляване

**++** – увеличава стойността на променливата с 1

**--** – намалява стойността на променливата с 1

| действие | нова стойност на **nc** |
|---|---|
| `nc = 5;` | 5 |
| `++nc;` | 6 |
| `nc++;` | 7 |
| | |
| `--nc;` | 6 |
| `nc--;` | 5 |

# Пример

```
k = 5;
z = k++;

(k = 6, z = 5)
```

```
k = 5;
z = ++k;

(k = 6, z = 6)
```

```
arr[i] = i++; //ГРЕШНО !!!

(a + j)++; //ГРЕШНО !!! - ++ и -- само на променливи!
```

# Масиви

```
тип име[размер];
```

тип – тип на масива
име – име на променливата
размер – брой елементи на масива, елементите се броят от **0** до **размер-1**:

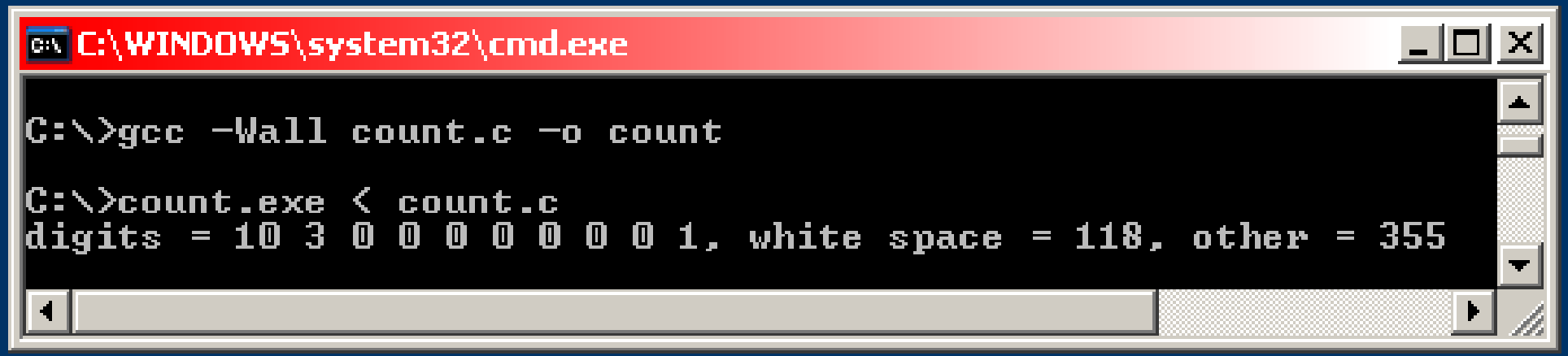
```
int test[3];
```

съществуват:
**test[0]**
**test[1]**
**test[2]**

# Пример$_1$

```
#include <stdio.h>
/* count digits, white space, others */
int main() {
    int c, i, nwhite, nother;
    int ndigit[10];
    nwhite = nother = 0;
    for (i = 0; i < 10; ++i)
       ndigit[i] = 0;
    while ((c = getchar()) != EOF)
       if (c >= '0' && c <= '9')
          ++ndigit[c-'0'];
       else if (c == ' ' || c == '\n' || c == '\t')
          ++nwhite;
       else  ++nother;
    printf("digits =");
    for (i = 0; i < 10; ++i)
       printf(" %d", ndigit[i]);
    printf(", white space = %d, other = %d\n", nwhite,
  nother);
}
```

# Пример 2


C:\WINDOWS\system32\cmd.exe
C:\>gcc -Wall count.c -o count
C:\>count.exe < count.c
digits = 10 3 0 0 0 0 0 0 0 1, white space = 118, other = 355

# Функции

```
return_type function_name(arguments) {
    /* function body */
}
```

- Аргументите (тип и име), ако ги има, се изреждат разделени със запетайки.
- Връщаната стойност може да бъде и **`void`**, т.е. функцията не връща стойност.
- Стойност се връща чрез оператор **`return;`** При неговото изпълнение се излиза незабавно от функцията, ако не се срещне **`return;`** върнатата стойност е случайна.

# Пример$_1$

```
#include <stdio.h>
int power(int m, int n);

/* test power function */
int main() {
     int i;
     for (i = 0; i < 10; ++i)
        printf("%d %d %d\n", i, power(2,i), power(-3,i));
     return 0;
}

/* power: raise base to n-th power; n >= 0 */
int power(int base, int n) {
     int i, p;
     p = 1;
     for (i = 1; i <= n; ++i)
        p = p * base;
     return p;
}
```

# Пример$_2$

```
0 1 1
1 2 -3
2 4 9
3 8 -27
4 16 81
5 32 -243
6 64 729
7 128 -2187
8 256 6561
9 512 -19683
```

# Аргументи на функции

- Предават се само по стойност, т.е. създава се копие на променливата и функцията работи с това копие, а не с оригинала
- Когато аргумент на функция е масив, то той не се копира! Подава се адресът на първия (нулев) елемент и промените се извършват директно върху оригинала
- За да може функция да променя променливите, подадени й като аргументи, а не техните копия, се използват указатели (**`pointers`**)

# Символни низове

- Низовете представляват масиви от символи:

```
char име[размер];
```

- Последният елемент на всеки низ е символ с ASCII код 0. По такъв начин се разбира и дължината на низа - броят се символите до първия срещнат нулев.

| индекс: | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| As char: | 's' | 't' | 'r' | 'i' | 'n' | 'g' | '\0' |

| индекс: | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| ASCII: | 115 | 116 | 114 | 105 | 110 | 103 | 0 |

# Пример$_1$

```
#include <stdio.h>
#define MAXLINE 1000 /* maximum input line length */

int getline(char line[], int maxline);
void copy(char to[], char from[]);

/* print the longest input line */
int main() {
     int len;              /* current line length */
     int max;              /* maximum length seen so far */
     char line[MAXLINE];   /* current input line */
     char longest[MAXLINE]; /* longest line saved here */
     max = 0;
     while ((len = getline(line, MAXLINE)) > 0)
        if (len > max) {
           max = len;
           copy(longest, line);
        }
     if (max > 0)     /* there was a line */
        printf("%s", longest);
     return 0;
}
```

# Пример 2

```
/* getline: read a line into s, return length */
int getline(char s[],int lim) {
     int c, i;
     for (i=0; i < lim-1 && (c=getchar())!=EOF && c!='\n'; ++i)
        s[i] = c;
     if (c == '\n') {
        s[i] = c;
        ++i;
     }
     s[i] = '\0';
     return i;
}

/* copy: copy 'from' into 'to'; assume to is big enough */
void copy(char to[], char from[]) {
     int i;
     i = 0;
     while ((to[i] = from[i]) != '\0')
        ++i;
}
```

# Външни променливи и област на действие

- Променливите декларирани в една функция са локални (частни) за нея
- Всяка локална променлива се създава само когато функцията бъде извикана и се унищожава, след като се излезе от функцията
- Всяка такава променлива трябва да бъде инициализирана, в противен случай не може да се предвиди точно нейното съдържание.

- В езика **C** е възможно да се дефинират външни (глобални) променливи
- Такива променливи се дефинират извън функциите и се декларират във функциите, които ще ги използват
- Декларирането става чрез оператора **`extern`**
- Операторът **`extern`** може да бъде пропуснат ако дефиницията на променливата се намира преди употребата й в дадена функция

# Пример

Файл 1 (main.c):

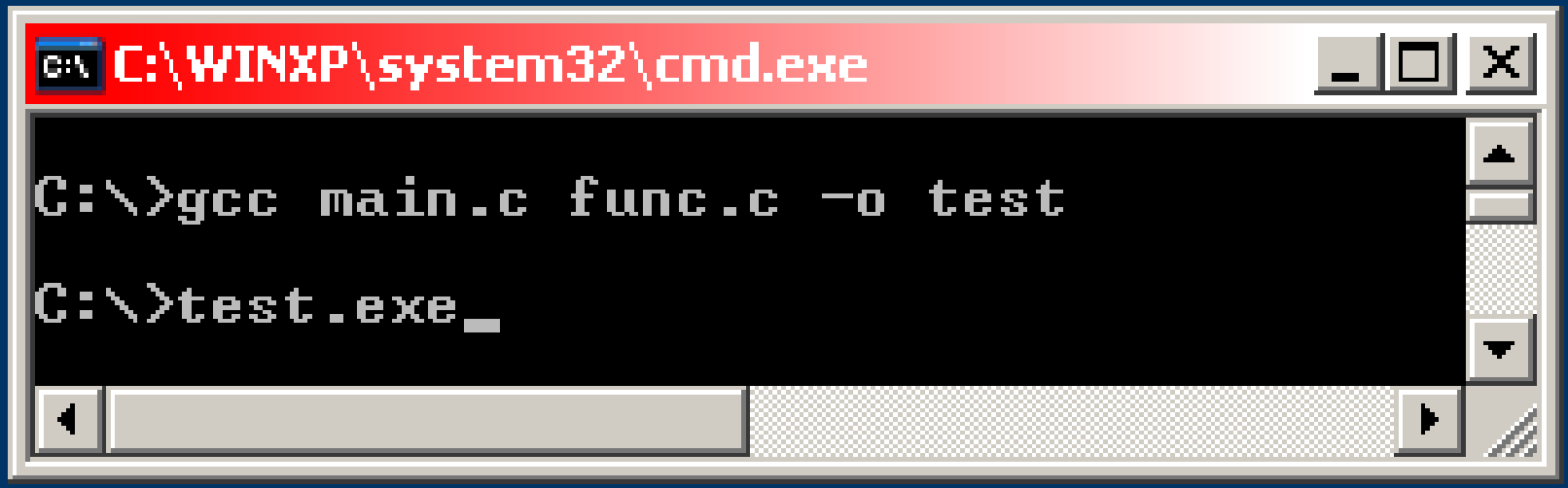
```
extern int i; //декларация

int main() {
 ++i;// i = 6 ...
}
```

Файл 2 (func.c):

```
/* дефиниция, тук се заделя
  място за променливата */

int i = 5;
```



# Пример

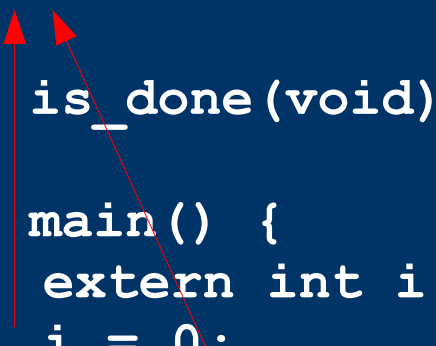
```
int i; //дефиниция, заделя се памет

int is_done(void);

int main() {
     extern int i;  //декларация
     i = 0;         //инициализация
     while (!is_done()) {
        i++;
        //do something
     }
}

int is_done(void) {
   extern int i; //декларация - ползва се вече дефинираното i

   return (i==6);
}
```



# Пример 1

```
#include <stdio.h>
#define MAXLINE 1000    /* максимална дължина на входния ред */

int max;                /* текуща максимална дължина */
char line[MAXLINE];     /* текущ входен ред */
char longest[MAXLINE]; /* най - дългия ред */

int getline(void);
void copy(void);
```

# Пример 2

```
/* отпечатва най - дългия входен ред*/
int main() {
     int len;               /* дължина на текущя ред */
     extern int max;
     extern char longest;

     max = 0;
     while ((len = getline()) > 0)
        if (len > max) {
           max = len;
           copy();
        }
     if (max > 0)       /* имало е ред на входа */
        printf("%s", longest);
     return 0;
}
```

# Пример 3

```
int getline() {
     int c, i;
     extern char line[];
     for (i=0; i < lim-1 && (c=getchar())!=EOF && c!='\n'; ++i)
        s[i] = c;
     if (c == '\n') {
        s[i] = c;
        ++i;
     }
     s[i] = '\0';
     return i;
}

void copy() {
     int i;
     extern char line[], longest[];
     i = 0;
     while ((to[i] = from[i]) != '\0')
        ++i;
}
```