ПРОГРАМНИЯТ ЕЗИК C

Ненко Табаков
Пламен Танов
Технологическо 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/

ИСТОРИЯ НА ЕЗИКА

- Създаден през 1978 година от Денис Ричи
- C 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

- **C** е език с общо предназначение
- В езика C са дефинирани фундаментални типове за цели числа, числа с плаваща запетая и символи
- Предоставя :
  - конструкции за управление на последователността на изпълнение
  - предпроцесорна обработка
  - възможност за създаване на собствени типове данни
  - адресна аритметика

ПРИМЕР

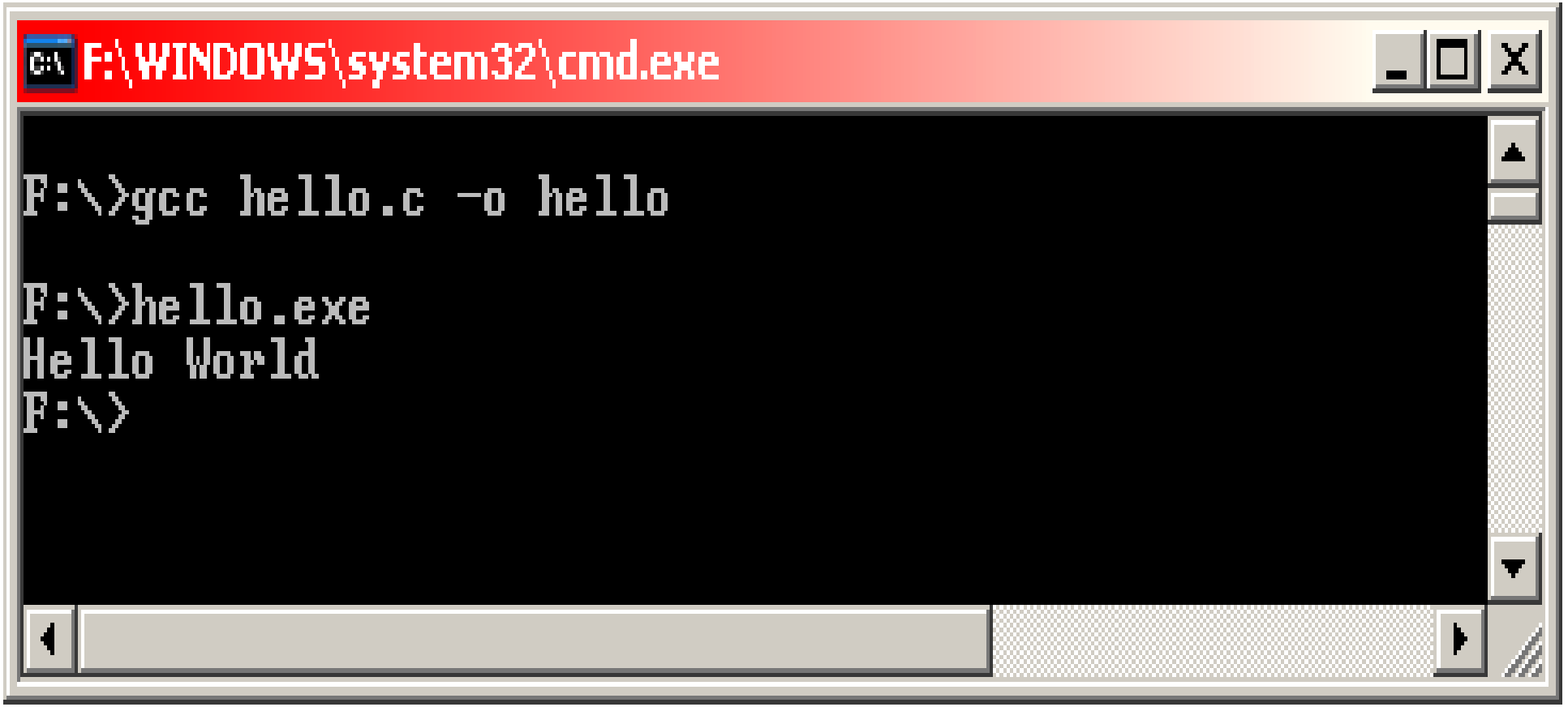
```
#include <stdio.h>

int main ()
{
   printf (“Hello world”);
}
```

включва информация за библиотеката

име на функцията
начало на функцията
извикване на функция
край на функцията

ИЗХОД НА ЕКРАНА



ПРИМЕР

```
#include <stdio.h>

/* отпечатва таблица по Фаренхайт и Целзий
1°C = (5/9) * (1°F - 32)*/

int main (){
   int fahr, celsius;
   int lower, upper, step;

   lower = 0;
   upper = 300;
   step = 20;

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

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	148

КОМЕНТАРИ В C

коментари в **C**

/* .... */

```
/*
   това е коментар
*/
```

ОСНОВНИ ТИПОВЕ В C

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

```
int fahr;
float celsius;
```

ЦИКЪЛЪТ `while`

while (*условие*)
{
  ...
}

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

ОПЕРАТОРЪТ `if`

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

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

ФУНКЦИЯТА `printf`

```
printf ("низ от символи", арг1, арг2, ...)
```

%d – изписва целочислена стойност
%f – изписва реална стойност
%c – изписва символ

\n – нов ред
\t - табулатор

```
printf ("%d\t%f\t\n", 5/9, 5.0/9.0);
```

ПРИМЕР

```
#include <stdio.h>

   int main ()
   {
      int fahr;

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

ЦИКЪЛЪТ `for`

for (*инициализация*;*условие; стъпка*)
{
  ...
}

```
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;
}
```

ДИРЕКТИВАТА **define**

#define *ИМЕ СТОЙНОСТ*

```
#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));
   }
```

ПРИМЕР

```
#include <stdio.h>

/* версия 1*/
int main ()
{
      int c;
      c = getchar ();
      while (c != EOF)
      {
         putchar (c);
         c = getchar ();
      }
}
```

```
#include <stdio.h>

/* версия 2*/
int main ()
{
      int c;
      while ((c = getchar ())!= EOF)
      {
         putchar (c);
      }
}
```

ИЗРАЗИ И ПРИОРИТЕТ НА ДЕЙСТВИЕ

- **=** – оператор за присвояване на стойност
- **!=** – логически оператор “различно от”
- **==** – логически оператор “еднакво на”

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

еквивалентно на

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

ПРИМЕР

```
#include <stdio.h>

int main ()
{
      int c;
      int nc = 0;
      while ((c = getchar ())!= EOF)
         ++nc;
      printf ("%d\n", nc);

}
```

ОПЕРАТРОРИ ЗА УВЕЛИЧАВАНЕ И НАМАЛЯВАНЕ

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

```
nc = 5;
++nc;
nc++;

--nc;
nc--;
```

стойността на nc е 5
стонността на nc е 6
стонността на nc е 7

стонността на nc е 6
стонността на nc е 5

МАСИВИ

***тип име*[*размер*];**

тип – тип на масива
име – име на променливата
размер – брой на елементите на масива. Елементите се броят от 0.

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

съществуват елементите:

```
test [0];
test [1];
test [2];
```

ПРИМЕР

```
#include <stdio.h>
/* брои цифри, празни места и други */
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);
}
```

ФУНКЦИИ

```
тип-на-върнатата-стойност име-на-функцията (аргументи){
...
}
```

- 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

ПРИМЕР

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

/* тества функцията power*/
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: повдига base на n-та степен; n >= 0 */
int power(int base, int n) {
   int i, p;
   p = 1;
   for (i = 1; i <= n; ++i)
      p = p * base;
   return p;
}
```

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.

ПРИМЕР

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

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

/* отпечатва най – дългия входен ред*/
int main() {
    int len;              /* дължина на текущя ред */
    int max;              /* текуща максимална дължина */
    char line[MAXLINE];     /* текущ входен ред */
    char longest[MAXLINE]; /* най – дългия ред */
    max = 0;
    while ((len = getline(line, MAXLINE)) > 0)
        if (len > max) {
            max = len;
            copy(longest, line);
        }
    if (max > 0)     /* имало е ред на входа */
        printf("%s", longest);
    return 0;
}
```

ПРИМЕР

```
/* getline: чете ред в s, връща дължината му */
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: копира 'from' в 'to'; предполагаме, че to е достатъчно
голям */
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 ще съдържа някакви ненужни данни

- 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 ѝ в дадена функция

ПРИМЕР

```
#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

```
/* 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

```
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;
}
```