Lecture 1 - PRELIMINARIES

The C character set

C uses the upper case letters A to Z, lower case letters a to z, the digits 0 to 9, and certain special characters as building blocks to form basic program elements such as constants, variables, operators expressions.

The list of special characters

```
!   *   +   \   "   <
*   (   =   |   {   >
%   )   ~   ;   }   /
^   -   [   :   ,   ?
&   _   ]   '   .   (blank)
```

Special sequences – build as the combinations, e.g. `\b \n \t` and so on.

Identifiers and keywords

Identifiers are names given to various program elements, such as variables, functions and arrays.

Identifiers consist of letters, digits and eventually the underscore sign (_) (treated as a letter) in any order, except that the first character must be a letter.

Examples of valid identifiers:

```
x      y12      sum_1     _temp
names  area     tax_rate  TABLE
```

Keywords – certain reserved words that have standard predefined meanings in C.

Standard keywords:

```
auto      extern     sizeof
break     float      static
case      for        struct
char      goto       switch
const     if         typedef
continue  int        union
default   long       unsigned
do        register   void
double    return     volatile
else      short      while
enum      signed
```

Some compilers recognize some or all of the following keywords:

```
ada       far        near
asm       fortran    pascal
entry     huge
```

Data types

Typical data types

```
Data type    Description            Typical memory
                                    requirements
int          integer quantity       2 bytes or 1 word
                                    (varies from one
                                    compiler to another)

char         single character       1 byte

float        floating point number  1 word (4 bytes)

double       double precision floa- 2 words (8 bytes),
             ting point number (i.e.
             more significant fi-
             gures, and an exponent
             which may be larger
             in magnitude)
```

Constants

Example	Name
1234	int
2345l	long int
2345L	long int
1234u	unsigned int
2345U	unsigned int
5678ul	unsigned long int
12.34	double
1e-2	double
1.2e-2	double
2.3f	float
2.3F	float
3.4e-2l	long double
3.4e-3L	long double

Different number systems

Number system		
decimal	octal	hexadecimal
31	037	0x1f
31	037	0X1F

Character constants

A character constant is a single character, enclosed in apstrophes.

```
'A'    'a'   '3'   '$'   ' '
```

American Standard Code for Information Interchange (ASCII) character set

```
  Constant        Value
     'A'           65
     'x'           120
     '3'           51
     '$'           36
     ' '           32
```

Escape sequences

```
Character        Escape sequence      ASCII value

bell(alert)           \a                 007
backspace             \b                 008
horizontal tab        \t                 009
vertical tab          \v                 011
newline (line feed)   \n                 010
form feed             \f                 012
carriage return       \r                 013
quotation mark        \"                 034
apostrophe            \'                 039
question mark         \?                 063
backslash             \\                 092
null                  \0                 000
octal escape seq.     \ooo
hexadec. esc seq.     \xoo
```

Constant expression – expression containing only constants

```
#define MAXL 10000
#define VTAB '\013'
#define VTAB '\x7'
#define VTAB '\X7'
#define VTAB '\v'
```

String constants – string of characters enclosed in double quottaion marks

```
"This is a string!"
or
"" /* empty string*/
```

Technically string is a table with a number of elements greater of one than the number of charaters included. The charaters are followed by the null character (\0).

Constant 'x' is not equal to "x".

Character strings may be sticked together during compilation of the program:

```
"Hey!" "Adventure!"
is identical with:
"Hey!Adventure!"
```

Enumeration constants

(List of values of integer constants)

```
enum boolean {NO,YES};
enum escapes {BELL='\a', BACKSPACE='\b', TAB='\t',
              NEWLINE='\n', VTAB='\v', RETURN='\r'};

enum months {JAN=1, FEB, MAR, APR, MAY, JUN, JUL,
             AUG, SEP, OCT, NOV, DEC};
   /*  months: february is second, march third
       and so on  */
```

Names in different enumerations must be different. At the same enumeration values may be repeated.

Types name *enum* shares the same space as the names of structure and union types.

Names of enumeration variables belongs to the same class as the identifiers of the ordinary variables.

Declarations

```
int lower,upper, step;
char c,lin[1000];
```

```
int lower;
int upper;
int step;
char c;
char lin[1000];
```

Initial values can be assigned to variables within a type declaration:

```
char esc='\\';   /*  \   character    */
int  i=0;        /*  iterations counter  */
int  limit=MAXLINE+1; /* maximal number of iterations*/
float eps=1.0e-5;     /* accuracy parameter         */
```

Default initial values of variables:

```
static and outer  -- 0
automatic         -- when the initial values are
                     explicitly stated the same
                     value with call of the
                     function or entry to the
                     block.
                     without explicitly stated
                     initial value they have
                     random values.
```

Qualifier *const* (constant) (may be used to declare any variable) It says that its value shall not be changed.

```
const double e=2.1234e-2;
const char msg[]="Uwaga";
int strlen(const char []);
/* table contents -- argument can not be
      changed inside the function     */
```

Any trial to change value of a variable declared as constant is ended in the way depending on an implementation.

```
char text[]="California";
```

An 11 - element array.

```
char text[11]="California";
```

Size should be specified correctly.

```
char text[6]="California";  /*  end will be lost  */
char text[20]="California"; /*  extra array elements
             may be assigned zeros, or may be filled
             with meaningless characters  */
```

Expressions

An expression represents a single data item, such as a number or a character. The expression may consist of a single entity, such as a constant, a variable, an array element or a reference to a function. It may also consist of some combinations of such elements interconnected by one or more operators.

Expressions can also represent logical conditions (in C true is represented by integer 1, false by 0).

Statements

A statement causes computer to carry out some action. There are three different classes of statements in C: *expression statements, compound statements and control statements*.

Expression statement – expression followed by a semicolon (;).

```
a=3;
c=a+b;
++i;
printf("Area = %f", area);
```

Compound statement – several individual statements enclosed within a pair of braces ({ and }

```
{
  pi=3.141593;
  circum=2. * pi * radius;
  area = pi * radius * radius;
}
```

Control statements

```
while (count <= n)  {
     printf("x= ");
     scanf("%f",&x);
     sum += x;
     ++count;
}
```