2. DATA TYPES

Identifiers constitute only the first part of a variable declaration. The other part specifies the type of data the identifier is to hold. **There are three basic data types in C**: integers, reals, and characters.

2.1 Basic data types

Characters

We usually think of characters as letters of the alphabet, but they encompass more than that. There are characters for digits and punctuation, as well as to represent special actions such as ringing a bell or causing form feed.

Internally, every character is represented by a small integer. What characters are available characters and how they are represented internally depends on the machine on which the program runs. The most common character sets are ASCII (American Standard Code for Information Interchange). ASCII is the character set used on most personal, micro, and minicomputers, as well as several large mainframes.

Type	Size in bits	Range
char	8	-127 to 127
unsigned char	8	0 to 255
signed char	8	-127 to 127

Integers

Integers are whole numbers. C provides three classes of integer storage: *short*, *int*, and *long*, in both *signed* and *unsigned* forms. These classes vary in size and efficiency of use. The following table lists the sizes and ranges of these values for Turbo C. On most systems, a *short* is 16 bits, an *int* is either 16 or 32 bits, and a *long* 32 bits. But the only guarantee is that a *short* is not going to be larger than an *int*, which is not going to be larger than a *long*.

Why do we have all of these different classes? Because they all have different uses. *int*s occupy one word of storage and are generally the most efficient data type to use. But since the word size varies between machines, we use *long*s for values that require more than 16 bits. *short*s allow us to save space when a variable's value always falls within a small range.

Integers can be either signed or unsigned. Signed integers use one bit for the sign of the number. A *signed int* (or just *int*, the default) uses one bit for the sign and 15 bits for the magnitude on 16-bit machines or 31 bits for the magnitude on 32-bit machines. Unsigned integers have no sign bit and use all the bits for the magnitude, which doubles the range of values they can represent. When a variable will only hold nonnegative values, as in a loop counter or an array index, making it unsigned increases the range of values it can hold. We declare unsigned integers with *unsigned int* (or simply, *unsigned*), *unsigned short*, and *unsigned long*.

Type	Size in bits	Range
int	16	-32,767 to 32,767
signed int	16	-32,767 to 32,767
unsigned int	16	0 to 65,535
short int	16	-32,767 to 32,767
signed short int	16	-32,767 to 32,767
unsigned short int	16	0 to 65,535
long int	32	-2,147,483,647 to 2,147,483,647
signed long int	32	-2,147,483,647 to 2,147,483,647
unsigned long int	32	0 to 4,294,967,295

Reals

Reals, or floating point numbers, are stored differently than integers. Internally, they are broken into a fraction and an exponent. The numbers of bits for each is machine-dependent, but a typical representation for a 32-bit real uses 23 bits for the fraction and 9 bits, including the sign, for the exponent, as shown below. Bit 1 is the sign of the fraction, bits 2 through 9 are the exponent, and bits 10 through 32 contain the fraction.

1	2 9	10 32
SIGN	EXPONENT	FRACTION

We normally write real numbers with a decimal point, as in 12.5 0r -13.45, but we can also write them in '**e**' notation, giving both a fraction and an exponent to base 10. '**e**' notation is similar to scientific notation, except that the letter '**e**' replaces the times sign and the base.

Floating Point	'e' notation	Scientific Notation
12.45	1.245e1	1.245×10^{1}
-211.0	-2.110e2	-2.110×10^{2}
0.0056	5.600e-3	5.600×10^{-3}
-0.000123	-1.230e-4	-1.230×10^{-4}
1000000.0	1e6	1.0×10^{6}

There are two types of real values, ***float*** and ***double***. A *float* requires 32 bits of storage and provides around seven significant digits. A *double* requires 64 bits of storage and provides approximately 14 significant digits. There is a third type, *long double*, which is designed to provide even more significant digits, but in Turbo C a *long double* is no different from a *double*. *double*s provide more significance and a wider range of values than *float*s, but require more space.

Type	Size in bits	Range
float	32	6 digits precision
double	64	10 digits precision
long double	128	10 digits precision

2.2 Identifiers and Naming Conventions

Variable and function names are known as **identifiers**. But not every combination of characters is a legal identifier. First, identifiers consist only of lowercase and uppercase letters, digits, and underscore (_) characters - no other characters are allowed. And second, the first character must be a letter or an underscore. Because some system functions begin with an underscore, however, always starting identifiers with a letter will prevent name conflicts.

Case is significant in identifiers: **sum** refers to a different identifier than **Sum**, which refers to a different identifier than **SUM**.

Turbo C reserves the identifiers listed below as keywords, so we can't use them as normal identifiers (the identifiers we've labeled with an asterisk are extensions to the standard C keywords; use the **OPTIONS / COMPILER / SOURCE** menu entry to allow these keywords to be used as identifiers). Turbo C also reserves a set of identifiers beginning with tc, so avoid starting names with this prefix. Finally, other compilers reserve *entry* and *fortran*, so avoid using them as well.

We can have identifiers of any length, but most compilers limit the number of significant characters. In the original definition of C only the first 8 characters were significant, implying that *var_name1* and *var_name2* referred to the same identifier. In Turbo C, as in most modern compilers, the first 32 characters are significant, although to be portable to all current compilers, it may be necessary to restrict names to six, seven, or eight characters. You can use the **OPTIONS / ENVIRONMENT** menu entry to specify the desired number of significant characters.

It's wise to choose variable names that are related to the purpose of the variable, and that are unlikely to get mixed up typographically. Pick names that reflect the intended use of the variable. The reasons are obvious. Just as with the comment statement, meaningful variable names can dramatically increase the readability of a program and will pay off in the debug and documentation phases. In fact, the documentation task will probably be greatly reduced since the program will be more self-explanatory. We tend to use short names for local variables, especially loop indices, and longer names for external variables.

Turbo C reserved words

asm*	auto	break	case	cdecl*	char
const	continue	default	do	double	else
enum	extern	far*	float	for	goto
huge*	if	int	interrupt*	long	near*
pascal*	register	return	short	signed	sizeof

static	struct	switch	typedef	union	unsigned
void	volatile	while			

2.3 Constants

Numeric Constants

Constants are unchangeable quantities. In C, an integer constants are expressed as a string of digits, with a leading minus sign indicating a negative number (eg. 123, 45, 12500, -35). A number that is too large for an *int* is automatically treated as a *long* (eg. 170000). To force a number to be treated as a *long* (that is, stored using 32 bits), place the letter 'l' or 'L' after it. 123L is a *long* constant. To force a number to be treated as *unsigned*, place the letter 'u' or 'U' after it. So, 123U is an *unsigned* constant. The suffixes can be combined to create *unsigned long* constants as in 123LU. There are no short constants.

In addition to decimal numbers, the value of an integer can be specified in octal or hexadecimal instead of decimal. A leading 0 (zero) on an integer constant means octal; a leading 0x or 0X means hexadecimal. For example, decimal 31 can be written as 037 in octal and 0x1f or 0X1F in hexadecimal. Regardless of how the value is specified, it is stored in its binary equivalent. Octal and hexadecimal constants may also be followed by L to make them *long* and U to make them unsigned: 0XFUL is an *unsigned long* constant with value 15 decimal.

By default, any constant we define in either floating point or exponential notation is a *double* (eg. 45.6). To have *float* constants, we follow the number with an 'f' or 'F', as in 45.6F. 45.6L is a long double.

Character Constants

A **character constant** is an integer, written as one character within single quotes, such as 'x'. The value of a character constant is the numeric value of the character in the machine's character set. For example, in the ASCII character set the character constant '0' has the value 48, which is unrelated to the numeric value 0. If we write '0' instead of a numeric value like 48 that depends on character set, the program is independent of the particular value and easier to read. Character constants participate in numeric operations just as any other integers, although they are most often used in comparisons with other characters.

Certain character cannot be typed between quotes like above.(eg. the carriage return). We represent these characters by an escape sequence: a backslash (\ ) followed by a code letter for the particular function. These sequences look like two characters, but represent only one. In addition, an arbitrary byte-sized bit pattern can be specified in octal or hexadecimal.

Characters available using the backslash quoting mechanism

Code	Character	Hex Value
\ 0	Null character	0
\ a	Audible bell	0x07
\ b	Backspace	0x08
\ f	Form feed	0x0C
\ n	New line	0x0A
\ r	Carriage return	0x0D
\ t	Horizontal tab	0x09
\ v	Vertical tab	0x0B
\ ‘	Single quote	0x2C
\ “	Double quote	0x22
\ ?	Question mark	0x3F
\ \	Backslash	0x5C
\ ddd	Up to 3-digit octal value	
\ xddd	Up to 3-digit hexadecimal value	

String Constants

A **string constant** is a sequence of zero or more characters surrounded by double quotes, as in “Hello World” or “ ” (blank space) the empty string. The double quotes are not part of the string, but serve only to delimit it. The same escape sequences used in character constants apply in strings; \ “ represents the double-quote character.

“a” - is a string
‘a’ - is a character

Technically, a string constant is an array of characters. The internal representation of a string has a **null character** (‘ **\0** ’) at the end, so the physical storage required is one more than the number of characters written between the quotes. This representation means that there is no limit to how long a string can be, but programs must scan a string completely to determine its length.

2.4 Declarations

In C, unlike some programming languages, we have to declare the type of every variable before use. The variables are declared by giving their type and name:

```
type    variable-list;

eg.     int  a, b, c ;
        float  d, f ;
```

Declarations specify the **storage class**, **type**, **name** and **optional initialization** of an object.

Storage classes : **auto**, **static**, **register**, **extern**