Data Types:

Character Constants

A *character constant* is a single character enclosed in single or double quotes. MASM stores the value in memory as the character's binary ASCII code. Examples are

```
'A'
"d"
```

String Constants

A *string constant* is a sequence of characters (including spaces) enclosed in single or double quotes:

```
'ABC'
'X'
"Good night, Gracie"
'4096'
```

Embedded quotes are permitted when used in the manner shown by the following examples:

```
"This isn't a test"
'Say "Good night," Gracie'
```

Reserved Words

Reserved words have special meaning in MASM and can only be used in their correct context. There are different types of reserved words:

- Instruction mnemonics, such as MOV, ADD, and MUL
- Register names
- Directives, which tell MASM how to assemble programs
- Attributes, which provide size and usage information for variables and operands. Examples are BYTE and WORD
- Operators, used in constant expressions
- Predefined symbols, such as @data, which return constant integer values at assembly time

Type	Usage
BYTE	8-bit unsigned integer. B stands for byte
SBYTE	8-bit signed integer. S stands for signed
WORD	16-bit unsigned integer (can also be a Near pointer in real-address mode)
SWORD	16-bit signed integer
DWORD	32-bit unsigned integer (can also be a Near pointer in protected mode). D stands for double
SDWORD	32-bit signed integer. SD stands for signed double
FWORD	48-bit integer (Far pointer in protected mode)
QWORD	64-bit integer. Q stands for quad
TBYTE	80-bit (10-byte) integer. T stands for Ten-byte
REAL4	32-bit (4-byte) IEEE short real
REAL8	64-bit (8-byte) IEEE long real
REAL10	80-bit (10-byte) IEEE extended real

Defining BYTE and SBYTE Data

The BYTE (define byte) and SBYTE (define signed byte) directives allocate storage for one or more unsigned or signed values. Each initializer must fit into 8 bits of storage. For example,

```
value1 BYTE 'A'          ; character constant
value2 BYTE 0            ; smallest unsigned byte
value3 BYTE 255          ; largest unsigned byte
value4 SBYTE –128        ; smallest signed byte
value5 SBYTE +127        ; largest signed byte
```

A question mark (?) initializer leaves the variable uninitialized, implying it will be assigned a value at runtime:

```
value6 BYTE ?
```

The optional name is a label marking the variable's offset from the beginning of its enclosing segment. For example, if **value1** is located at offset 0000 in the data segment and consumes 1 byte of storage, **value2** is automatically located at offset 0001:

```
value1 BYTE 10h
value2 BYTE 20h
```

The DB directive can also define an 8-bit variable, signed or unsigned:

```
val1 DB 255           ; unsigned byte
val2 DB -128          ; signed byte
```

Defining Strings

To define a string of characters, enclose them in single or double quotation marks. The most common type of string ends with a null byte (containing 0). Called a *null-terminated* string, strings of this type are used in many programming languages:

```
greeting1 BYTE "Good afternoon",0
greeting2 BYTE 'Good night',0
```

Each character uses a byte of storage. Strings are an exception to the rule that byte values must be separated by commas. Without that exception, **greeting1** would have to be defined as

```
greeting1 BYTE 'G','o','o','d' ....etc.
```

which would be exceedingly tedious. A string can be divided between multiple lines without having to supply a label for each line:

```
greeting1 BYTE "Welcome to the Encryption Demo program "
BYTE "created by Kip Irvine.",0dh,0ah
BYTE "If you wish to modify this program, please "
BYTE "send me a copy.",0dh,0ah,0
```

The hexadecimal codes 0Dh and 0Ah are alternately called CR/LF (carriage-return line-feed) or *end-of-*

line characters. When written to standard output, they move the cursor to the left column of the line following the current line.

The line continuation character (\) concatenates two source code lines into a single statement. It must be the last character on the line. The following statements are equivalent:

```
greeting1 BYTE "Welcome to the Encryption Demo program "
```

and

```
greeting1 \
BYTE "Welcome to the Encryption Demo program "
```

Defining WORD and SWORD Data

The WORD (define word) and SWORD (define signed word) directives create storage for one or more 16-bit integers:

```
word1 WORD 65535          ; largest unsigned value
word2 SWORD -32768        ; smallest signed value
word3 WORD ?              ; uninitialized, unsigned
```

The legacy DW directive can also be used:

```
val1 DW 65535          ; unsigned
val2 DW -32768         ; signed
```

Defining DWORD and SDWORD Data

The DWORD (define doubleword) and SDWORD (define signed doubleword) directives allocate storage for one or more 32-bit integers:

```
val1 DWORD 12345678h       ; unsigned
val2 SDWORD –2147483648    ; signed
val3 DWORD 20 DUP(?)       ; unsigned array
```

The legacy DD directive can also be used:

```
val1 DD 12345678h        ; unsigned
val2 DD –2147483648      ; signed
```

The DWORD can be used to declare a variable that contains the 32-bit offset of another variable. Below, **pVal** contains the offset of **val3**:

```
pVal DWORD val3
```

Defining QWORD Data

The QWORD (define quadword) directive allocates storage for 64-bit (8-byte) values:

```
quad1 QWORD 1234567812345678h
```

The legacy DQ directive can also be used:

```
quad1 DQ 1234567812345678h
```

Similarly, rest of the data types can be used.

1. Division Operation:

Mnemonics	Syntax	Working	Example

div	`div op1`	Divide value of `eax` register with op1 and quotient goes to `eax` and remainder goes to `edx`.	`mov b, 5`
			`mov eax, 10`
			`div b`
			10 will be divided by 5 and quotient 2 will be stored in eax and remainder 0 in edx.

Task 1: *Ask user to enter two number and divide first number with second and print quotient and remainder.*

2. writestring:

This is used to print string on console. To do that, offset of string is copied to `edx` register. As a string is a consecutive array of character stored in consecutive memory locations (bytes), we first move offset of first byte to `edx` (`mov edx, offset string; considering string a variable`) then the writestring will print all characters of strings.

Task 2: *Declare a variable, initialize it with any string and print it on console.*