**Chap 2. Basic computation**

**Variables and Data Types**

A variable is a name referencing to a **memory location** (**address**)

* Holds binary data
* 2 basic operations: **set** value, **get** value.
* When the program is compiled, the **compiler will determine** the **position** where the variable is allocated

The operator **&** will get the address of a variable or code.

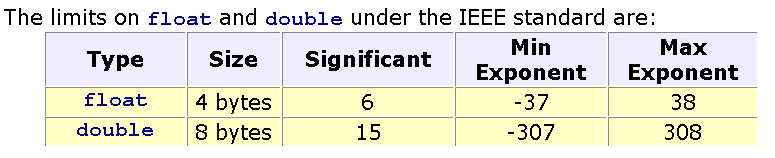
The operator **sizeof**(var/type) return the size (number of byte) occupied by a variable/type.

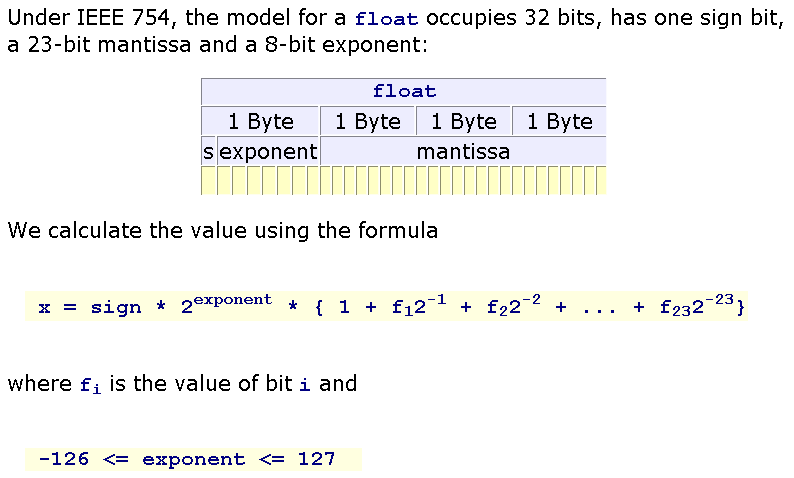
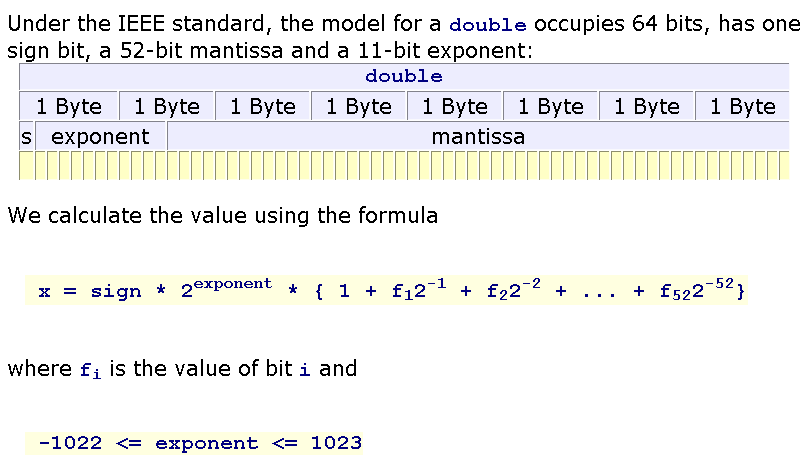
***Qualifiers:***

* We can qualify the int data type so that it contains a minimum number of bits.
* Qualifiers:
  + **short** :at least 16 bits
  + **long**: at least 32 bits
  + **long** **long**: at least 64 bits

***Cultural Symbols (characters):***

* Using an integral data type.
* Store a symbol by storing the **integer associated with the symbol**.
* Over 60 encoding sequences have already been defined.

**Floating-Point Data:** 2 components: an exponent, a mantissa (phần định trị)

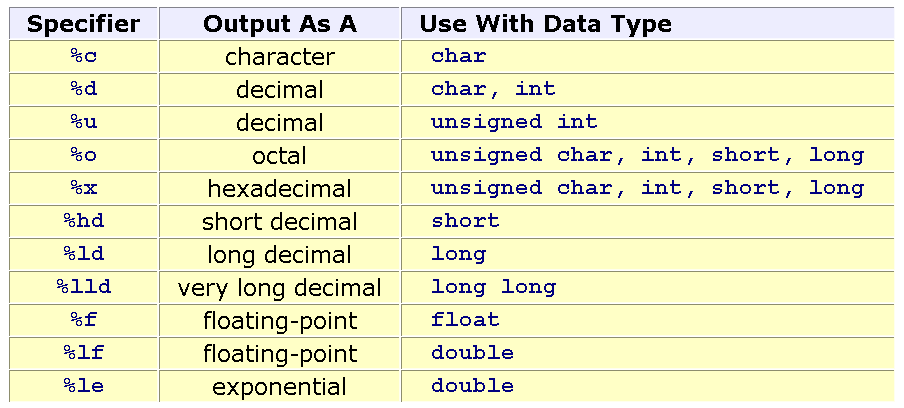


**Literals**

Constant values are specified directly in the source code.

* **Character** literals (constant characters)
* **String** literals(constant strings)
* **Number** literals (constant numbers)

**Conversion Specifiers**

****

**Expressions**

* Arithmetic operators
* Relational operators
* Logical operators
* Bit operators
* Shorthand Assignment Operators
* Casting
* Precedence

**Chap 3. Basic logic**

**Logic constructs:** Expressions enable us to write programs that **perform calculations** **and execute statements** in a sequential order.

**Structured Programming**

Structure of a program code should be organize in a manner so that it is **understandable**, **testable** and **readily** **modifiable**.

DESIGN using: **pseudo-coding** or **flow charting**

Logic constructs:

* Sequence constructs: imple statements, code blocks
* Selection constructs: ½ \_\_\_ if, else ; 1/n \_\_\_ switch | (condition) ? True\_Value : False\_Value
* Iteration constructs
  + - **Flags:** allow jumps across statements: goto, continue, and break
    - A **flag** is a variable that keeps track of a true or false state

**Walkthroughs**

* + **A record of the changes** that occur in the values of program variables as a program executes
  + A listing of the output, if any, produced by the program.

Ways to perform a walkthrough

* + Memory Map
* Walkthrough Tables

**Chap 4. Module and Functions**

**Modules** – design units

**Define:** A portion of a program that carries out a specific small function and may be used alone or combined with other modules to create a program.

**Characteristics**

* It is easy to upgrade and maintain
* It can be re-used in the same program: has an identified name
* It can be re-used in some programs: restore in a library

**Identifying:**

* One entry point and one exit point,
* Highly cohesive: code of a module focuses to the determined purpose and some related modules are put in one file.
* Low coupling: modules are independent.

**C-Functions**

* **Void**: A function that does not return a value \_\_\_ a subroutine or procedure.
* **Main:** to which the operating system transfers control at the start of execution; entry point of a C prog.

**Parameters:** names of data in function implementation

**Arguments:** data used when a function is called

**Coercion:** ép kiểu dữ liệu

**Prototypes**

Function prototypes describe the form of a function without specifying the implementation details 🡪 Function declaration is put at a place and it’s implementation is put at other.

**The #include Directive:** lệnh #include

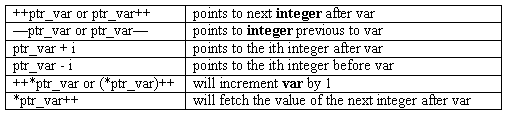
instruct the compiler to insert a copy of the header file into our source code.

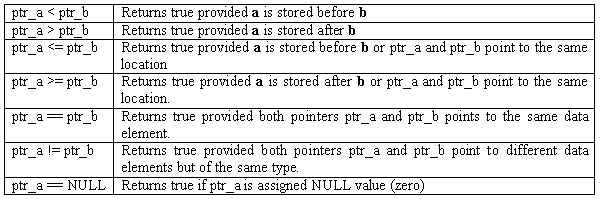
Syntax:  #include "filename" //in user directory

#include <filename> // in system directory

**Chap 5. Pointer**

A **pointer** is a variable, which **contains the address** of a **memory location** or **another variable.**

**Pointer Arithmetic Operator**

**Pointer Comparisons**

**Dynamic Allocated Data**

**Stdlib.h**

* void\* calloc (size\_t numberOfItem, size\_t bytesPerItem)
* void\* malloc (size\_t numBytes) ;
* void\* realloc (void\* curPointer, size\_t newNumBytes);
* void free(void\* willBeDeletedPointer);

**Size\_t**: Another name of the int type. It is used in case of memory allocation managing.

**void** is the general datatype which means that the data type is not determined yet. So, user must give an explicit casting when it is used.

e.g:

**int** n; **int** \*p1;

p1 = (int\*)malloc(sizeof(int));

free(p1);

double \*p1, \*p2;

p1 = (double\*) malloc ( sizeof(double));

p2 = (double\*) malloc ( sizeof(double));

**Chap 6. Library**

The standard C libraries: Standard, Time, Math, Character

**The Standard library: Stdlib.h**

**The Time library: time.h**

* **CLOCKS\_PER\_SEC**: Number of clock ticks per second. A constant is defined in *time.h* (CLOCKS\_PER\_SEC = 1000 means that 1 clock tick = 1milisecond.
* 2 data type are defined in the library *time.h:*
* **time\_t:**  typedef long time\_t;
* **clock\_t:** typedef long clock\_t;

**time\_t time ( time\_t \*tptr );** returns the current calendar time and this time is stored in it’s parameter.

**double difftime ( time\_t , time\_t );** returns the difference in seconds between two calendar time arguments.

**clock\_t clock ( void );** returns the current date time information using the unit clock tick

**The Math Library: Math.h**

**INPUT**

To transfer the contents of a buffer to a program the user must press the '\n' character.

**Stream**: A concept allows generalizing access data in a buffer as a chain of characters.

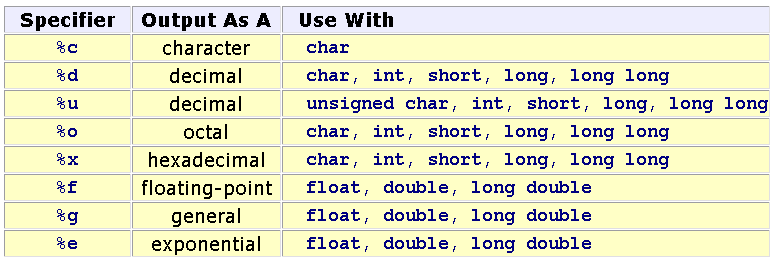
C functions provide buffered input facilities on the standard input stream:

* **getchar() :** get a character from the keyboard
* **getchar()** retrieves a single character from the standard input stream buffer without translating the input. Prototype: **int getchar(void );**
* **getchar** returns either the character code / **EOF**.
* (EOF=-1, ctrl+z in Windows, ctrl+d in Unix)
* Clear buffer: *fflush(stdin)*
* **scanf (…) :** get data from the keyboard
* **scanf** retrieves data values from the standard input stream buffer under format control.**Syntax:** scanf( format string, &identifier , ... )
* **scanf** extracts data values from the standard input stream buffer until scanf has
* interpreted and processed the entire format string,
* found a character that does not meet the next conversion specification
* Whitespace: **newline, horizontal tab, form feed, vertical tab and space characters**.

**Input Validation**

* invalid characters
* trailing characters
* out-of-range input
* incorrect number of input fields

**Formatted Output**

* ****putchar(int) : character
* printf ( format\_string, varList): list of data

**Chap 7. Cont Storage**

**1-D Arrays: Declaration**

float \*a;

a = (float\*)calloc (10, sizeof(float)); /\* allocate a block of 10 float numbers \*/

**How to access the ith element of the array a?**

***a*** is the address of the first element. Based on operation on pointers:

* + ***a+i*** : address of the ith element, another way: ***&a[i]***
  + ***\*(a+i)***: value of the ith element, another way: ***a[i]***

**Dynamic**: double \*a = (double\*)calloc(n, sizeof(double));

**2-D Arrays**

Keep in your mind the way to specify a matrix as a parameter of a function ( the number of column must be pre-defined.).

**Static arrays** 🡪 Stack segment

Type a[MAXN];

Type m[MAXROW][MAXCOL];

**Dynamic array:** Use pointer and allocate memory using functions

int\*\* m = (int\*\*) calloc(row, sizeof(int\*));

for (i=0; i<row; i++) m[i]= (int\*)calloc(col, sizeof(int));

|  |  |  |  |
| --- | --- | --- | --- |
| **1-D Array (a)** | | **2-D Array (m)** | |
| ***Address*** | ***Value*** | ***Address*** | ***Value*** |
| &a[index] | a[index] | &m[i][j] | m[i][j] |
| a+index | \*(a+index) |  |  |
| **Compiler determines the address of an element:** | | | |
| a + index\*sizeof(DataType) | | m + (i\*NumCol + j)\*sizeof(DataType) | |

**How to accept blanks in a input string?**

* **%[^\n]** conversion specifier

**Chap 10. File**

The fundamental unit of a file is a **byte**.

A file is a stream of **bytes**.

A file concludes with a special mark called the end of file mark (EOF).

* Text file: Unit of data in a file is an ASCII code of character.
* Binary file: Unit of data in a binary byte 🡪 Each byte on the file is a direct image of the corresponding byte in memory

**Declaration**

FILE\* identifier: FILE\* f=NULL;

**The fopen() function**

**Text file:**

* **"r"** - read from the file,
* **"w"** - write to the file:
* **"a"** - write to the end of the file
* **"r+"** - opens the file for reading and possibly writing,
* **"w+"** - opens the file for writing and possibly reading;
* **"a+"** - opens the file for writing to the end and possibly reading

**Binary files:** *“rb”, “wb”, “r+b”, “w+b”,”a+b”*