UNIX and POSIX Standards

The ANSI C Standard

ANSIC Standard X3.159-1989

The difference between ANSI C AND K&R C

- **Function prototyping:**

  *** ANSI C :**

  data-type function-name (data type parameter name,……………………….)

  Ex: int f1(int a , int b);

* K&R C :

data-type function-name (parameter
name,……………………………..)

```
EX: int f1(a , b);
    int a, b;
```

- Constant and volatile qualifiers

  * Present in ANSI C not in K&R C

  * const-implies data cant be changed

```
    /*here printf cant change the
                                    value of x  */
     int printf(const char* x,….)
       {
       }
```

- Volatile qualifier : implies the compiler can make any optimization of the variable

EX :
```
char get_io()
        {
          volatile char* io_port=0x7777;
           char ch=*io_port;
           ch = *io_port;
         }
```

- Wide character support and internationalization

  *support to store characters which occupy more than one byte

  *ANSI C defines SETLOCALE function

  *which helps to specify the format of date

  monetary and real number presentation

SETLOCALE

```
#include <locale.h>
Char setlocale (int category, const char* locale);
```

Category 1	*Category 2*
LC_TYPE	en_US//US
LC_MONETARY	fr_FR//French
LC_NUMERIC	de_DE//German
LC_TIME	
LC_ALL	

- Permit function pointers to be used without dereferencing

*ANSI C –a function pointer can be used like a function

*K&R C – requires the pointer to be de referenced to call the function

Feature test macros

STDC	: 1-if underlying system is ANSI C compliant 0-Otherwise
LINE	: Physical line number of the module
FILE	: filename of module where the symbol is present
DATE	: date of compilation of the module
TIME	: time of compilation of the module

```
#include <stdio.h>
    int main()
      {
        #if __STDC__ == 0 && !defined(__cplusplus)
              printf("cc is not ANSI C compliant\n");
         #else
         printf(" %s compiled at %s:%s. This statement is
                       at   line    %d\n",
              __FILE__, __DATE__, __TIME__, __LINE__);
         #endif
                       return 0;
     }
```

THE ANSI/ISO C++STANDARD

- WG21-ISO and ANSI X3J16 : ANSI C/ISO
  C++ standard

Version 3.0 report : c++ should have

* classes
* derived classes
* virtual classes
* operator overloading
* template classes
* template function
* exception handling
* io stream

ANSI C AND ANSI C++

ANSI C	ANSI C++
-default prototype if called before declaration or defn	- prototype is mandatory
-int f1() is same as int f1(…)	-int f1() is same as int f1(void)
-no type safe linkage	-type safe linkage

THE POSIX STANDARDS

Posix.1 : IEEE 1003.1-1990 adapted by ISO as ISO/IEC 9945:1:1990 standard

***gives standard for base operating system API**

Posix.1b : IEEE 1003.4-1993

*** gives standard APIs for real time operating system interface including interprocess communication**

- **Posix.1c** ： **specifies multi thread programming interface**

Other POSIX compliant systems

***VMS of DEC**

***OS/2 of IBM**

***W-NT of Microsoft**

***Sun solaris 2.t**

***HP-UX 9.05**

- To ensure program confirms to POSIX.1 standard user should define _POSIX_SOURCE as

1. #define _POSIX_SOURCE OR
2. Specify -D _POSIX_SOURCE to a C++ compiler

_POSIX_C_SOURCE : its value indicating POSIX version

- *_POSIX_C_SOURCE* **value----Meaning**

198808L**----**	**First version of POSIX.1 compliance**
199009L----	**Second version of POSIX.1 compliance**
199309L----	**POSIX.1 and POSIX.1b compliance**

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include <iostream.h>
#include <unistd.h>
int main()
{
    #ifdef _POSIX_VERSION
        cout << "System conforms to POSIX: " <<
    _POSIX_VERSION << endl;
    #else
       cout << "_POSIX_VERSION is undefined\n";
    #endif
    return 0;
}
```

POSIX ENVIRONMENT

- **Difference between POSIX and UNIX**

*** In UNIX C and C++ header files are included in /usr/include**

In POSIX they are just headers not header files and /usr/include neednot exist

*** UNIX – Superuser has special previlege and the superuser ID is always 0**

POSIX – Doesnot support the concept of superuser nor the ID is 0

THE POSIX FEATURE TEST MACROS

- **_POSIX_JOB_CONTROL— The system supports BSD type job control**
- **_POSIX_SAVED_ID — keeps saved set-UID and set-GID**
- **_POSIX_CHOWN_RESTRICTED — If -1 user may change ownership of files owned by them else only users with special privilege can do so**

- _POSIX_NO_TRUNC — If -1 then any long path name is automatically truncated to NAME_MAX else an error is generated
- _POSIX_VDISABLE — If -1 then there is no dissabling character for special characters for all terminal devices otherwise the value is the disabling character value

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

int main()
{
#ifdef _POSIX_JOB_CONTROL
   cout << "System supports job control\n";
#else
   cout << "System does not support job control\n";
#endif
```

```
#ifdef _POSIX_SAVED_IDS

   cout << "System supports saved set-UID and saved
                set-GID\n";
#else

       cout << "System does not support saved set-UID
                 and saved set-GID\n";
#endif
```

```
#ifdef _POSIX_CHOWN_RESTRICTED

    cout << "chown restricted option is: " <<
         _POSIX_CHOWN_RESTRICTED <<endl;
#else

     cout << "System does not support system-wide
              chown_restricted option\n";
#endif
```

```
#ifdef _POSIX_NO_TRUNC

    cout << "Pathname trucnation option is: " <<
              _POSIX_NO_TRUNC << endl;
#else

   cout << "System does not support system-wide
              pathname trucnation option\n";

#endif

}
```

```
#ifdef _POSIX_VDISABLE

      cout << "Diable character for terminal files is: "
              << _POSIX_VDISABLE << endl;
#else

    cout << "System does not support
                 _POSIX_VDISABLE\n";
#endif

    return 0;
```

Certain constants defined in <limit.h>

- _POSIX_CHILD_MAX 6

  max number of child processes that can be created at any one time by a process
- _POSIX_OPEN_MAX 16

  max number of files that can be opened simultaneously by a process
- _POSIX_STREAM_MAX 8

  max number of I/Ostreams that can be opened simultaneously by a process

- _POSIX_ARG_MAX 4096

  max size, in bytes of arguments that can be passed to an exec function call
- _POSIX_NGROUP_MAX 0

  max number of supplemental groups to which a process may belong
- _POSIX_PATH_MAX 255

  max number of characters allowed in a pathname

- **_POSIX_NAME_MAX** **14**

  max number of characters allowed in a filename
- **_POSIX_LINK_MAX** **8**

  max number of links a file may have
- **_POSIX_PIPE_BUF** **512**

  max size of block of data that can be automatically read from or written to a pipe file

_POSIX_MAX_INPUT max capacity, in bytes, of a terminal's input queue	255
_POSIX_MAX_CANON max capacity, in bytes, of a terminal's canonical input queue	255
_POSIX_SSIZE_MAX max value that can be stored in a ssize_t- typed object	32767
_POSIX_TZNAME_MAX max number of characters in a time zone name	3

- Long sysconf(const int limit_name);
- Long pathconf(const char* pathname,int flimit_name);
- Long fpathconf(const int fdesc,int flimitname);

- Int res;
- If(res=sysconf(_SC_OPEN_MAX))==-1)
- perror(“sysconf”);
- Else cout<<res;
- res=pathconf(“/”,_PC_PATH_MAX);
- Res=fpathconf(0,_PC_CHOWN_RESTRICTED);

THE POSIX.1 FIPS STANDARD

- **Job control :**

  _POSIX_JOB_CONTROL must be defined
- **Saved set-UID and set-GID :**

  _POSIX_SAVED_IDS must be defined
- **Long path name is supported**

  _POSIX_NO_TRUNC != -1
- **_only authorised user can change ownership**

  _POSIX_CHOWN_RESTRICTED != -1

- _POSIX_VDISABLE should be defined
- NGROUP_MAX – value should be at least 8
- Read and write APIs should return the number of bytes transferred after the APIs have been interrupted by signals
- The group id of newly created file must inherit group ID of its containing directory

THE X/OPEN STANDARDS

- X/Open portability guide, ISSUE 3 (XPG3)

  --- 1989
- X/Open portability guide, ISSUE 4 (XPG4)

  --- 1999
- The portability guide specifies a set of common facilities and C application program interface function to be provided on all UNIX-based “open systems”

QUESTIONS

- What are the major differences between ANSI C and K & R C? explain (10)
- What is POSIX standard? Give the structure of the program to filter out non-POSIX compliant codes for a user program (10)
- What is an API ? How are they different from C library functions ? Calling an API is more time consuming than calling a user function . Justify or contradict (5)

- Write a POSIX compliant C/C++ program to check following limits (10)

1. Maximum path length
2. Maximum characters in a file name
3. Maximum number of open files per process

- What is POSIX standard? Explain different subsets of POSIX standard .write the structure of the program to filter out

  non-POSIX compliant codes for a user program (6)

- Write a C++ program that prints the POSIX defined configuration options supported on any given system using feature test macros (8)
- List out all POSIX.1 and POSIX 1b defined system configuration limits in manifested constants with complete time limit , minimum value and meaning (10)