MODULE I: INTRODUCTION

UNIX AND ANSI STANDARDS

UNIX is a computer operating system originally developed in 1969 by a group of AT&T employees at Bell Labs, including **Ken Thompson, Dennis Ritchie, Douglas McElroy and Joe Ossanna.** Today UNIX systems are split into various branches, developed over time by AT&T as well as various commercial vendors and non-profit organizations.

The ANSI C Standard

In 1989, **American National Standard Institute** (ANSI) proposed C programming language standard X3.159-1989 to standardise the language constructs and libraries. This is termed as ANSI C standard. This attempt to unify the implementation of the C language supported on all computer system.

The major differences between ANSI C and K&R C [Kernighan and Ritchie] are as follows:

- Function prototyping
- Support of the const and volatile data type qualifiers.
- Support wide characters and internationalization.
- Permit function pointers to be used without dereferencing.

Function prototyping

ANSI C adopts C++ function prototype technique where function definition and declaration include function names, arguments' data types, and return value data types. This enables ANSI C compilers to check for function calls in user programs that pass invalid number of arguments or incompatible arguments' data type.

These fix a major weakness of K&R C compilers: invalid function calls in user programs often pass compilation but cause programs to crash when they are executed.

```
Eg:     unsigned long foo(char * fmt, double data)
        {
                /*body of foo*/
        }
```

External declaration of this function foo is

```
        unsigned long foo(char * fmt, double data);

eg:     int printf(const char* fmt,.....);
```

specify variable number of arguments

Support of the const and volatile data type qualifiers.

- The **const** keyword declares that some data cannot be changed.
  Eg: `int printf(const char* fmt,.....);`
  Declares a fmt argument that is of a const char * data type, meaning that the function printf cannot modify data in any character array that is passed as an actual argument value to fmt.
- **Volatile** keyword specifies that the values of some variables may change asynchronously, giving an hint to the compiler's optimization algorithm not to remove any "redundant" statements that involve "volatile" objects.

```
eg:   char get_io()
      {
           volatile char* io_port = 0x7777;
           char ch = *io_port;        /*read first byte of data*/
           ch = *io_port;                   /*read second byte of data*/
      }
```

If io_port variable is not declared to be volatile when the program is compiled, the compiler may eliminate second ch = *io_port statement, as it is considered redundant with respect to the previous statement.

- The **const** and **volatile** data type qualifiers are also supported in C++.

Support wide characters and internationalisation

- ANSI C supports internationalisation by allowing C-program to use wide characters. Wide characters use more than one byte of storage per character.
- ANSI C defines the **setlocale** function, which allows users to specify the format of date, monetary and real number representations.
  For eg: most countries display the date in dd/mm/yyyy format whereas US displays it in mm/dd/yyyy format.
- Function prototype of setlocale function is:

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

- The setlocale function prototype and possible values of the category argument are declared in the <locale.h> header. The category values specify what format class(es) is to be changed.
- Some of the possible values of the category argument are:

category value		effect on standard C functions/macros
LC_CTYPE	⇒	Affects behavior of the <ctype.h> macros
LC_TIME	⇒	Affects date and time format.
LC_NUMERIC	⇒	Affects number representation format
LC_MONETARY	⇒	Affects monetary values format
LC_ALL	⇒	combines the affect of all above

Permit function pointers without dereferencing

ANSI C specifies that a function pointer may be used like a function name. No referencing is needed when calling a function whose address is contained in the pointer.

For Example, the following statement given below defines a function pointer funptr, which contains the address of the function foo.

```
extern void foo(double xyz,const int *ptr);
void (*funptr)(double,const int *)=foo;
```

The function foo may be invoked by either directly calling foo or via the funptr.

```
foo(12.78,"Hello world");
funptr(12.78,"Hello world");
```

K&R C requires funptr be dereferenced to call foo.

```
(* funptr) (13.48,"Hello usp");
```

ANSI C also defines a set of C processor(cpp) symbols, which may be used in user programs. These symbols are assigned actual values at compilation time.

cpp SYMBOL	USE
STDC	Feature test macro. Value is 1 if a compiler is ANSI C, 0 otherwise
LINE	Evaluated to the physical line number of a source file.
FILE	Value is the file name of a module that contains this symbol.
DATE	Value is the date that a module containing this symbol is compiled.
TIME	value is the time that a module containing this symbol is compiled.

The following **test_ansi_c.c** program illustrates the use of these symbols:

```
#include<stdio.h>
int main()
{
        #if _STDC_==0
                printf("cc is not ANSI C compliant");
        #else
                printf("%s compiled at %s:%s.
                        This statement is at line %d\n",
                        _FILE_ , _DATE_ , _TIME_ , _LINE_ );
        #endif
                Return 0;
}
```

- Finally, ANSI C defines a set of standard library function & associated headers. These headers are the subset of the C libraries available on most system that implement K&R C.

The ANSI/ISO C++ Standard

These compilers support C++ classes, derived classes, virtual functions, operator overloading. Furthermore, they should also support template classes, template functions, exception handling and the iostream library classes.

Differences between ANSI C and C++

ANSI C	C++
Uses K&R C default function declaration for any functions that are referred before their declaration in the program.	Requires that all functions must be declared / defined before they can be referenced.
int foo(); ANSI C treats this as old C function declaration & interprets it as declared in following manner. int foo(.....); → meaning that foo may be called with any number of arguments.	int foo(); C++ treats this as int foo(void); Meaning that foo may not accept any arguments.
Does not employ type_safe linkage technique and does not catch user errors.	Encrypts external function names for type_safe linkage. Thus reports any user errors.

The POSIX standards

- POSIX or "Portable Operating System Interface" is the name of a family of related standards specified by the IEEE to define the application-programming interface (API), along with shell and utilities interface for the software compatible with variants of the UNIX operating system.
- Because many versions of UNIX exist today and each of them provides its own set of API functions, it is difficult for system developers to create applications that can be easily ported to different versions of UNIX.
- Some of the subgroups of POSIX are POSIX.1, POSIX.1b & POSIX.1c are concerned with the development of set of standards for system developers.
- **POSIX.1**
  - This committee proposes a standard for a base operating system API; this standard specifies APIs for the manipulating of files and processes.
  - It is formally known as IEEE standard 1003.1-1990 and it was also adopted by the ISO as the international standard ISO/IEC 9945:1:1990.
- **POSIX.1b**
  - This committee proposes a set of standard APIs for a real time OS interface; these include IPC (inter-process communication).
  - This standard is formally known as IEEE standard 1003.4-1993.
- **POSIX.1c**
  - This standard specifies multi-threaded programming interface. This is the newest POSIX standard.
  - These standards are proposed for a generic OS that is not necessarily be UNIX system.
  - E.g.: VMS from Digital Equipment Corporation, OS/2 from IBM, & Windows NT from Microsoft Corporation are POSIX-compliant, yet they are not UNIX systems.
  - To ensure a user program conforms to POSIX.1 standard, the user should either define the manifested constant _POSIX_SOURCE at the beginning of each source module of the program (before inclusion of any header) as;

    ```
    #define _POSIX_SOURCE
    ```

    Or specify the -D_POSIX_SOURCE option to a C++ compiler (CC) in a compilation;

    ```
    %       CC -D_POSIX_SOURCE *.C
    ```
  - POSIX.1b defines different manifested constant to check conformance of user program to that standard. The new macro is _POSIX_C_SOURCE and its value indicates POSIX version to which a user program conforms. Its value can be:

_POSIX_C_SOURCE VALUES	MEANING
198808L	First version of POSIX.1 compliance
199009L	Second version of POSIX.1 compliance
199309L	POSIX.1 and POSIX.1b compliance

- _POSIX_C_SOURCE may be used in place of _POSIX_SOURCE. However, some systems that support POSIX.1 only may not accept the _POSIX_C_SOURCE definition.
- There is also a _POSIX_VERSION constant defined in <unistd.h> header. It contains the POSIX version to which the system conforms.

Program to check and display _POSIX_VERSION constant of the system on which it is run

```
#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 undefined\n";
#endif
       return 0;
}
```

The POSIX Environment

Although POSIX was developed on UNIX, a POSIX complaint system is not necessarily a UNIX system. A few UNIX conventions have different meanings according to the POSIX standards. Most C and C++ header files are stored under the /usr/include directory in any UNIX system and each of them is referenced by

#include<header-file-name>

This method is adopted in POSIX. There need not be a physical file of that name existing on a POSIX conforming system.

The POSIX Feature Test Macros

POSIX.1 defines a set of feature test macro's which if defined on a system, means that the system has implemented the corresponding features. All these test macros are defined in **<unistd.h>** header.

Feature test macro	Effects if defined
_POSIX_JOB_CONTROL	The system supports the BSD style job control.
_POSIX_SAVED_IDS	Each process running on the system keeps the saved set UID and the set-GID, so that they can change its effective user-ID and group-ID to those values via seteuid and setegid API's.
_POSIX_CHOWN_RESTRICTED	If the defined value is -1, users may change ownership of files owned by them, otherwise only users with special privilege may change ownership of any file on the system.
_POSIX_NO_TRUNC	If the defined value is -1, any long pathname passed to an API is silently truncated to NAME_MAX bytes, otherwise error is generated.
_POSIX_VDISABLE	If defined value is -1, there is no disabling character for special characters for all terminal device files. Otherwise the value is the disabling character value.

Program to print POSIX defined configuration options supported on any given system.

```
/* show_test_macros.C */
#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";
#else
       cout<<" system does not support job control\n";
#endif

#ifdef _POSIX_SAVED_IDS
       cout<<" system supports saved set-UID and set-GID";
#else
       cout<<" system does not support set-uid and gid\n";
#endif

#ifdef _POSIX_CHOWN_RESTRICTED
       cout<<"chown_restricted option is :"
              <<_POSIX_CHOWN_RESTRICTED<<endl;
#else
       cout<<"system does not support"
              <<" chown_restricted option\n";
#endif

#ifdef _POSIX_NO_TRUNC
       cout<<"pathname trunc option is:"
              << _POSIX_NO_TRUNC<<endl;
#else
       cout<<" system does not support system-wide pathname"
      <<"trunc option\n";
#endif

#ifdef _POSIX_VDISABLE
       cout<<"disable char. for terminal files is:"
              <<_POSIX_VDISABLE<<endl;
#else
       cout<<" system does not support _POSIX_VDISABLE \n";
#endif
       return 0;
}
```

Limits checking at Compile time and at Run time

POSIX.1 and POSIX.1b defines a set of system configuration limits in the form of manifested constants in the <limits.h> header.

The following is a list of POSIX.1 – defined constants in the <limits.h> header.

Compile time limit	Min. Value	Meaning
_POSIX_CHILD_MAX	6	Maximum number of child processes that may be created at any one time by a process.
_POSIX_OPEN_MAX	16	Maximum number of files that a process can open simultaneously.
_POSIX_STREAM_MAX	8	Maximum number of I/O streams opened by a process

		simultaneously.
_POSIX_ARG_MAX	4096	Maximum size, in bytes of arguments that may be passed to an *exec* function.
_POSIX_NGROUP_MAX	0	Maximum number of supplemental groups to which a process may belong
_POSIX_PATH_MAX	255	Maximum number of characters allowed in a path name
_POSIX_NAME_MAX	14	Maximum number of characters allowed in a file name
_POSIX_LINK_MAX	8	Maximum number of links a file may have
_POSIX_PIPE_BUF	512	Maximum size of a block of data that may be atomically read from or written to a pipe
_POSIX_MAX_INPUT	255	Maximum capacity of a terminal's input queue (bytes)
_POSIX_MAX_CANON	255	Maximum size of a terminal's canonical input queue
_POSIX_SSIZE_MAX	32767	Maximum value that can be stored in a ssize_t-typed object
_POSIX_TZNAME_MAX	3	Maximum number of characters in a time zone name

The following is a list of POSIX.1b – defined constants:

Compile time limit	Min. Value	Meaning
_POSIX_AIO_MAX	1	Number of simultaneous asynchronous I/O.
_POSIX_AIO_LISTIO_MAX	2	Maximum number of operations in one listio.
_POSIX_TIMER_MAX	32	Maximum number of timers that can be used simultaneously by a process.
_POSIX_DELAYTIMER_MAX	32	Maximum number of overruns allowed per timer.
_POSIX_MQ_OPEN_MAX	2	Maximum number of message queues that may be accessed simultaneously per process
_POSIX_MQ_PRIO_MAX	2	Maximum number of message priorities that can be assigned to the messages
_POSIX_RTSIG_MAX	8	Maximum number of real time signals.
_POSIX_SIGQUEUE_MAX	32	Maximum number of real time signals that a process may queue at any time.
_POSIX_SEM_NSEMS_MAX	256	Maximum number of semaphores that may be used simultaneously per process.
_POSIX_SEM_VALUE_MAX	32767	Maximum value that may be assigned to a semaphore.

Prototypes:

```
#include<unistd.h>

long sysconf(const int limit_name);
long pathconf(const char *pathname, int flimit_name);
long fpathconf(const int fd, int flimit_name);
```

The *limit_name* argument value is a manifested constant as defined in the <unistd.h> header. The possible values and the corresponding data returned by the *sysconf* function are:

Limit value	*Sysconf* return data
_SC_ARG_MAX	Maximum size of argument values (in bytes) that may be passed to an exec API call
_SC_CHILD_MAX	Maximum number of child processes that may be owned by a process simultaneously
_SC_OPEN_MAX	Maximum number of opened files per process

_SC_NGROUPS_MAX	Maximum number of supplemental groups per process
_SC_CLK_TCK	The number of clock ticks per second
_SC_JOB_CONTROL	The _POSIX_JOB_CONTROL value
_SC_SAVED_IDS	The _POSIX_SAVED_IDS value
_SC_VERSION	The _POSIX_VERSION value
_SC_TIMERS	The _POSIX_TIMERS value
_SC_DELAYTIMERS_MAX	Maximum number of overruns allowed per timer
_SC_RTSIG_MAX	Maximum number of real time signals.
_SC_MQ_OPEN_MAX	Maximum number of messages queues per process.
_SC_MQ_PRIO_MAX	Maximum priority value assignable to a message
_SC_SEM_MSEMS_MAX	Maximum number of semaphores per process
_SC_SEM_VALUE_MAX	Maximum value assignable to a semaphore.
_SC_SIGQUEUE_MAX	Maximum number of real time signals that a process may queue at any one time
_SC_AIO_LISTIO_MAX	Maximum number of operations in one listio.
_SC_AIO_MAX	Number of simultaneous asynchronous I/O.

All constants used as a sysconf argument value have the _SC prefix. Similarly the flimit_name argument value is a manifested constant defined by the <unistd.h> header. These constants all have the _PC_ prefix.
Following is the list of some of the constants and their corresponding return values from either pathconf or fpathconf functions for a named file object.

Limit value	*Pathconf* return data
_PC_CHOWN_RESTRICTED	The _POSIX_CHOWN_RESTRICTED value
_PC_NO_TRUNC	Returns the _POSIX_NO_TRUNC value
_PC_VDISABLE	Returns the _POSIX_VDISABLE value
_PC_PATH_MAX	Maximum length of a pathname (in bytes)
_PC_NAME_MAX	Maximum length of a filename (in bytes)
_PC_LINK_MAX	Maximum number of links a file may have
_PC_PIPE_BUF	Maximum size of a block of data that may be read from or written to a pipe
_PC_MAX_CANON	maximum size of a terminal's canonical input queue
_PC_MAX_INPUT	Maximum capacity of a terminal's input queue.

These variables may be used at compile time, such as the following:

```
char pathname [ _POSIX_PATH_MAX + 1 ];
for (int i=0; i < _POSIX_OPEN_MAX; i++)
      close(i);          //close all file descriptors
```

The following *test_config.C* illustrates the use of *sysconf, pathcong and fpathconf:*

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

int main()
{
       int res;
       if((res=sysconf(_SC_OPEN_MAX))==-1)
              perror("sysconf");
       else
              cout<<"OPEN_MAX:"<<res<<endl;
```

```
        if((res=pathconf("/",_PC_PATH_MAX))==-1)
                perror("pathconf");
        else
                cout<<"max path name:"<<(res+1)<<endl;
        if((res=fpathconf(0,_PC_CHOWN_RESTRICTED))==-1)
                perror("fpathconf");
        else
                cout<<"chown_restricted for stdin:"<<res<<endl;
        return 0;
}
```

The POSIX.1 FIPS Standard

FIPS stands for Federal Information Processing Standard. The FIPS standard is a restriction of the POSIx.1 – 1988 standard, and it requires the following features to be implemented in all FIPS-conforming systems:

- Job control
- Saved set-UID and saved set-GID
- Long path name is not supported
- The _POSIX_CHOWN_RESTRICTED must be defined
- The _POSIX_VDISABLE symbol must be defined
- The NGROUP_MAX symbol's value must be at least 8
- The read and write API should return the number of bytes that have been transferred after the APIs have been interrupted by signals
- The group ID of a newly created file must inherit the group ID of its containing directory.

The FIPS standard is a more restrictive version of the POSIX.1 standard

The X/OPEN Standards

The X/Open organization was formed by a group of European companies to propose a common operating system interface for their computer systems. The portability guides specify a set of common facilities and C application program interface functions to be provided on all UNIX based open systems. In 1973, a group of computer vendors initiated a project called "*common open software environment*" (COSE). The goal of the project was to define a single UNIX programming interface specification that would be supported by all type vendors. The applications that conform to ANSI C and POSIX also conform to the X/Open standards but not necessarily vice-versa.

UNIX AND POSIX APIs

API → A set of application programming interface functions that can be called by user programs to perform system specific functions.

Most UNIX systems provide a common set of API's to perform the following functions.

- Determine the system configuration and user information.
- Files manipulation.
- Processes creation and control.
- Inter-process communication.
- Signals and daemons
- Network communication.

The POSIX APIs

In general POSIX API's uses and behaviours' are similar to those of Unix API's. However, user's programs should define the _POSIX_SOURCE or _POSIX_C_SOURCE in their programs to enable the POSIX API's declaration in header files that they include.

The UNIX and POSIX Development Environment

POSIX provides portability at the source level. This means that you transport your source program to the target machine, compile it with the standard C compiler using conforming headers and link it with the standard libraries.

Some commonly used POSIX.1 and UNIX API's are declared in <unistd.h> header. Most of POSIX.1, POSIX>1b and UNIX API object code is stored in the libc.a and lib.so libraries.

API Common Characteristics

- Many APIs returns an **integer value** which indicates the termination status of their execution
- API **return -1** to indicate the **execution has failed**, and the global variable **errno** is set with an error code.
- a user process may call **perror** function to print a diagnostic message of the failure to the std o/p, or it may call **strerror** function and gives it errno as the actual argument value; the strerror function returns a diagnostic message string and the user process may print that message in its preferred way
- the possible error status codes that may be assigned to errno by any API are defined in the <errno.h> header.

Following is a list of commonly occur error status codes and their meanings:

Error status code	Meaning
EACCESS	A process does not have access permission to perform an operation via a API.
EPERM	A API was aborted because the calling process does not have the superuser privilege.
ENOENT	An invalid filename was specified to an API.
BADF	A API was called with invalid file descriptor.
EINTR	A API execution was aborted due to a signal interruption
EAGAIN	A API was aborted because some system resource it requested was temporarily unavailable. The API should be called again later.
ENOMEM	A API was aborted because it could not allocate dynamic memory.
EIO	I/O error occurred in a API execution.
EPIPE	A API attempted to write data to a pipe which has no reader.
EFAULT	A API was passed an invalid address in one of its argument.
ENOEXEC	A API could not execute a program via one of the exec API
ECHILD	A process does not have any child process which it can wait on.

Questions appeared in previous semester exams:

1. What are the major differences between ANSI C and K&R C? Explain with examples.

[Jan 2019]

ANSI C	C++
Uses K&R C default function declaration for any functions that are referred before their declaration in the program.	Requires that all functions must be declared / defined before they can be referenced.
int foo();	int foo();
ANSI C treats this as old C function declaration &	C++ treats this as int foo(void);
interprets it as declared in following manner. int foo(.....); ? meaning that foo may be called	Meaning that foo may not accept any arguments.
with any number of arguments.	
Does not employ type_safe linkage technique and does not catch user errors.	Encrypts external function names for type_safe linkage. Thus reports any user errors.

2. What do you understand by feature test macros? List all five feature test macros along with their meanings. [July 2019]

POSIX.1 defines a set of feature test macro's which if defined on a system, means that the system has implemented the corresponding features. All these test macros are defined in **<unistd.h>** header.

Feature test macro	Effects if defined
_POSIX_JOB_CONTROL	The system supports the BSD style job control.
_POSIX_SAVED_IDS	Each process running on the system keeps the saved set UID and the set- GID, so that they can change its effective user-ID and group-ID to those values via seteuid and setegid API's.
_POSIX_CHOWN_RESTRICTED	If the defined value is -1, users may change ownership of files owned by them, otherwise only users with special privilege may change ownership of any file on the system.
_POSIX_NO_TRUNC	If the defined value is -1, any long pathname passed to an API is silently truncated to NAME_MAX bytes, otherwise error is generated.
_POSIX_VDISABLE	If defined value is -1, there is no disabling character for special characters for all terminal device files. Otherwise the value is the disabling character value.

3. What is error status code? List and explain meaning of any 4 error status codes. [Jan 2020]

   Error status codes indicate the termination status of API's execution. API **return -1** to indicate the **execution has failed**, and the global variable **errno** is set with an error code.

Error status code	Meaning
EACCESS	A process does not have access permission to perform an operation via a API.
EPERM	A API was aborted because the calling process does not have the superuser privilege.
ENOENT	An invalid filename was specified to an API.
BADF	A API was called with invalid file descriptor.

Mr. Naveen S Pagad

Asst.Prof, Dept. of ISE