# Chapter 1 : Introduction

* The difference between ANSI-C and K&R-C

(1) Function Prototyping : ANSI-C supports strong type checking. it enables the compiler to check for the function calls in the user prg that passes invalid no of arg, return type of the function. Whereas K&R-C compiler wont fix this error and it may lead to prg crash when they are executed

(2) Support of constant and volatile keyword : Constant keyword declares that some data cannot be changed. Volatile keyword specifies that value of some variables may change asynchronously. Not Supported in K&R-C

Ex: Volatile char *port = 0X7777;

char *ch = *port;

ch = *port;

LC_TYPE
LC_TIME
LC_NUMERIC

(3) Supports wide void character and Internationalisati

wide void character uses more that one byte of storag per character. ANSI-C support setlocale function which allows user to specify the format of date and set nos in different countries

char *setlocale (int category, const char *locati

↳ date, number ↓ which count

(4) Permits function pointers to be used it whil deferencing

void foo(double xyz, char *temp);
void (*funpt)(double, char *) = foo.

ANSI-C { foo(12.3, "Hi");
funpt(12.3, "Hi");

KfR { (*funpt)(12.3, "Hi");

ANSI-C specifies that a function ptr may be used as a function name. No dereferencing is need when calling a function whose addr is contained in the pointer.

* Difference Between ANSI-C and C++

| ANSI-C | C++ |
|---|---|
| (1) It uses KfR-C defaults. Function declaration for any function that are referenced before their declaration in the prg is accepted. | It requires that all functions must be declared or defined that can be referenced. |
| (2) void foo(); its equivalent in ANSI-C is void foo(...): This means the fun^n 'foo' can be called with any no. of actual arguments. | void foo(); its equivalent in C++ is: void foo(void): means that fun^n foo may not accept any arguments |
| (3) It does not employ type safe linkage. * ↳ invalid declared referenced | It encrypts external function names for type safe linkage. |

* ANSI-C also defines a set of C-preprocessor symbols which may be used in the user prg and are assigned values at compile time

The various symbols are:

__STDC__ : This macro is used as a test macro, that is, if its value is '1', it is a ANSI-C compiler if '0', some other compiler

__LINE__ : It displays the line number of a src file for which the symbol is referenced.

__FILE__ : It displays the file name that contains this symbol

__DATE__ : It specifies the date of when the file was compiled.

* POSIX STANDARDS:

Many versions of UNIX exits today and each of them provide its own set of API's, it is difficult for system developer to create the applications that can be easily ported on different version of UNIX

To overcome this the IEEE society found the special task force called POSIX to create a set of standards for operating system interface

POSIX.1

This committee proposed a set of standards for base OS API's which specifies for manipulation of files and processes.

POSIX.1b

This committee proposed a set of standard API's for real time OS which include IPC.

POSIX.1c.

This standard specifies multithreaded programming interface.

* POSIX's TEST MACROS:

- POSIX_JOB_CONTROL : This macro supports the Berkelyne Style Distribution. BSD style job control.
- POSIX_SAVED_ID : Here each process running on the system keeps the set_UID and set_GID so that it can change the effective UID and GID to those values via set_UID and set_GID.
- POSIX_CHOWN_RESTRICTED : If this macro value is -1 it changes the ownership of the file otherwise only privileged user may change the ownership of the file.
- POSIX_NO_TRUNC : If the macro value is -1 any long path name passed to it is truncated to name_max (NAME_MAX) bytes otherwise an error is generated
- POSIX_VDISABLE : If the macro value is -1, there is no disabling characters for special characters for all terminal device files.

* LIMIT CHECKING AT COMPILE TIME AND AT RUN TIME: To find actual implementation configuration we can use sysconf, pathconf and fpathconf functions at run-time

**sysconf** : This is used to query the system-wide configuration limits that are implemented on a given system

**pathconf** : It is used to query file related configuration limits where it take file pathname as argument

**fpathconf** : It is used to query file related configuration limits where it take file descriptors as its arguments.

```
#include <unistd.h>
        long sysconf(const int limit_name);
        long pathconf(const char *pathname,
                      const int limit_name);
        long fpathconf(const int fdes, const
                       int limit_name);
```

* 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 | Max. no. of child processes that may be created at any one time by a process |
| _POSIX_OPEN_MAX | 16 | Max. no. of files that may be opened simultaneously by a process. |
| _POSIX_PATH_MAX | 255 | Max no. of characters allowed in a file name |

| | | |
|---|---|---|
| _POSIX_NAME_MAX | 14 | Max. no of characters allowed in a file name. |
| _POSIX_LINK_MAX | 8 | Max no. of links a file may have. |
| _POSIX_STREAM_MAX | 8 | Max. no of I/O streams that may be opened simultaneously by a process |

* The followed is a list of POSIX.1b - defined constants:

| Compile time limit | Min. Value | Meaning |
|---|---|---|
| _POSIX_AIO_MAX | 1 | No. of simultaneously asynchronous I/O |
| _POSIX_AIO_LISTIO_MAX | 2 | Max no. of operations in one listio |
| _POSIX_TIMER_MAX | 32 | Max no. of timers that can be used simultaneously by a process. |
| _POSIX_DELAYTIME_MAX | 32 | Max no of overruns allowed per timer |
| _POSIX_RTSIG_MAX | 8 | Max no. of real time signals. |

* FIPS (Standards):

Federal Information Processing Standards

It is a guideline for standards. These guidelines have been extracted from POSIX.1 standards. If the system satisfies the

following features then it is said to be implemented with FIPS standards:

1. It should support Job Control
2. It should support set_UID and set_GID functions.
3. It should not support long pathnames
4. _POSIX_CHOWN_RESTRICTED must be defined explicitly.
5. _POSIX_VDISABLE must be defined.
6. READ and WRITE API should return the no. of bytes that have been texted, after API has been interrupted by signals.
7. GID of newly created file must inherit the GID of its containing directory:

* API Characteristics:

Most of the API's return an integer value which indicates the termination status of their execution. If an API return -1, it means that the API execution has failed and a global variable "errono" is set with an error code. The variable perror displays the message of the error code to the standard output or the log files.

The various error code status are:

EACCESS : A process does not have access permission to perform an operation.

EPERM : It means an API was aborted as the calling process does not have the superuser privileges.

BADF : It means an API was called with an invalid file descriptor.

ENOENT : It means an invalid filename was specified to an API.

EINTR : which means an API execution was aborted due to signal interruption

EAGAIN : This means an API was aborted because some of the system resources requested were temporarily unavailable.

ENOMEM: which means the API was aborted because it could not allocate dynamic memory.

EIO : An I/O error encountered in an API execution.

ECHILD : which means a process does not have any child process which it is waiting for.