INTRODUCTION

ENDECS

Basically ENDECS is a cryptography tool, which is used to compress and encrypt the original information into cipher text file to send the data from sender to the receiver side. At the receiver side the cipher text and the encryption key is used to get the plain text. This process is used to transfer the sensitive information from any medium (wireless or wired network). ENDECS is a console application which can be operated only in the terminal mode of operating system. This application is basically designed for GNU Linux operating systems. It is very fast in order to generate the cipher and plain text . Right now its operations are limited as you can see in the picture:

1. CRYPTOGRAPHY

Purpose

Cryptography is the use of codes to convert data so that only a specific recipient will be able to read it, using a key.

Cryptography is closely related to the disciplines of cryptology and cryptanalysis. Cryptography includes techniques such as microdots, merging words with images, and other ways to hide information in storage or transit. However, in today's computer - centric world, cryptography is most often associated with scrambling plaintext (ordinary text, sometimes referred to as clear text) into ciphertext (a process called encryption), then back again (known as decryption). Individuals who practice this field are known as cryptographers.

Modern cryptography concerns itself with the following four objectives:

1) **Confidentiality** (the information cannot be understood by anyone for whom it was unintended)

2) **Integrity** (the information cannot be altered in storage or transit between sender and intended receiver without the alteration being detected)

3) **Non-repudiation** (the creator/sender of the information cannot deny at a later stage his or her intentions in the creation or transmission of the information)

4) **Authentication** (the sender and receiver can confirm each other?s identity and the origin/destination of the information)

Procedures and protocols that meet some or all of the above criteria are known as cryptosystems. Cryptosystems are often thought to refer only to mathematical procedures and computer programs; however, they also include the regulation of human behavior, such as choosing hard-to-guess passwords, logging off unused systems, and not discussing sensitive procedures with outsiders.

The word is derived from the Greek *kryptos*, meaning hidden. The origin of cryptography is usually dated from about 2000 BC, with the Egyptian practice of hieroglyphics. These consisted of complex pictograms, the full meaning of which was only known to an elite few. The first known use of a modern cipher was by Julius Caesar (100 BC to 44 BC), who did not trust his messengers when communicating with his governors and officers. For this reason, he created a system in which each character in his messages was replaced by a character three positions ahead of it in the Roman alphabet.

In recent times, cryptography has turned into a battleground of some of the world's best mathematicians and computer scientists. The ability to securely store and transfer sensitive information has proved a critical factor in success in war and business.

Because governments do not wish certain entities in and out of their countries to have access to ways to receive and send hidden information that may be a threat to national interests, cryptography has been subject to various restrictions in many countries, ranging from limitations of the usage and export of software to the public dissemination of mathematical concepts that could be used to develop cryptosystems. However, the Internet has allowed the spread of powerful programs and, more importantly, the underlying techniques of cryptography, so that today many of the most advanced cryptosystems and ideas are now in the public domain.

1.1 ENCRYPTION

The primary purpose of encryption is to protect the confidentiality of digital data stored on computer systems or transmitted via the **Internet** or other computer **networks**. Modern encryption **algorithms** play a vital role in the security assurance of IT systems and communications as they can provide not only confidentiality, but also the following key elements of security:

- Authentication: the origin of a message can be verified.
- Integrity: proof that the contents of a message have not been changed since it was sent.
- Non-repudiation: the sender of a message cannot deny sending the message.

1.1.1 How we use encryption today

Until the arrival of the Diffie-Hellman key exchange and RSA algorithms, governments and their armies were the only real users of encryption. However, Diffie-Hellman and RSA led to the broad use of encryption in the commercial and consumer realms to protect data both while it is being sent across a network (data in transit) and stored, such as on a hard drive, smartphone or flash drive (data at rest). Devices like modems, set-top boxes, smartcards and SIM cards all use encryption or rely on protocols like SSH, S/MIME, and SSL/TLS to encrypt sensitive data. Encryption is used to protect data

in transit sent from all sorts of devices across all sorts of networks, not just the Internet; every time someone uses an ATM or buys something online with a smartphone, makes a mobile phone call or presses a key fob to unlock a car, encryption is used to protect the information being relayed. Digital rights management systems, which prevent unauthorized use or reproduction of copyrighted material, are yet another example of encryption protecting data.

Data, often referred to as plaintext, is encrypted using an encryption algorithm and an encryption key. This process generates ciphertext that can only be viewed in its original form if decrypted with the correct key. Decryption is simply the inverse of encryption, following the same steps but reversing the order in which the keys are applied. Today's encryption algorithms are divided into two categories: symmetric and asymmetric.

Symmetric-key ciphers use the same key, or secret, for encrypting and decrypting a message or file. The most widely used symmetric-key cipher is AES, which was created to protect government classified information. Symmetric-key encryption is much faster than asymmetric encryption, but the sender must exchange the key used to encrypt the data with the recipient before he or she can decrypt it. This requirement to securely distribute and manage large numbers of keys means most cryptographic processes use a symmetric algorithm to efficiently encrypt data, but use an asymmetric algorithm to exchange the secret key.

Asymmetric cryptography, also known as public-key cryptography, uses two different but mathematically linked keys, one public and one private. The public key can be shared with everyone, whereas the private key must be kept secret. RSA is the most widely used asymmetric algorithm, partly because both the public and the private keys can encrypt a message; the opposite key from the one used to encrypt a message is used to decrypt it. This attribute provides a method of assuring not only confidentiality, but also the integrity, authenticity and non-reputability of electronic communications and data at rest through the use of digital signatures.

1.1.2. Cryptographic hash functions

A cryptographic hash function plays a somewhat different role than other cryptographic algorithms. Hash functions are widely used in many aspects of security, such as digital signatures and data integrity checks. They take an electronic file, message or block of data and generate a short digital fingerprint of the content called a message digest or hash value. The key properties of a secure cryptographic hash function are:

- Output length is small compared to input
- Computation is fast and efficient for any input
- Any change to input affects lots of output bits
- One-way value-- the input cannot be determined from the output
- Strong collision resistance -- two different inputs can't create the same output

In 2012, the National Institute of Standards and Technology (NIST) announced Keccak as the winner of its Cryptographic Hash Algorithm Competition to select a next-generation cryptographic hash algorithm. The Keccak (pronounced "catch-ack") algorithm will be known as SHA-3 and complement the SHA-1 and SHA-2 algorithms specified in FIPS 180-4, Secure Hash Standard. Even though the competition was prompted by successful attacks on MD5 and SHA-0 and the emergence of theoretical attacks on SHA-1, NIST has said that SHA-2 is still "secure and suitable for general use."

The ciphers in hash functions are built for hashing: they use large keys and blocks, can efficiently change keys every block and have been designed and vetted for resistance to related-key attacks. General-purpose ciphers used for encryption tend to have different design goals. For example, the symmetric-key block cipher AES could also be used for generating hash values, but its key and block sizes make it nontrivial and inefficient.

1.2. DECRYPTION

Decryption is the process of taking encoded or encrypted text or other data and converting it back into text that you or the computer can read and understand. This term could be used to describe a method of un-encrypting the data manually or with un-encrypting the data using the proper codes or keys.

Decryption is generally the reverse process of encryption. It is the process of decoding the data which has been encrypted into a secret format. An authorized user can only decrypt data because decryption requires a secret key or password.

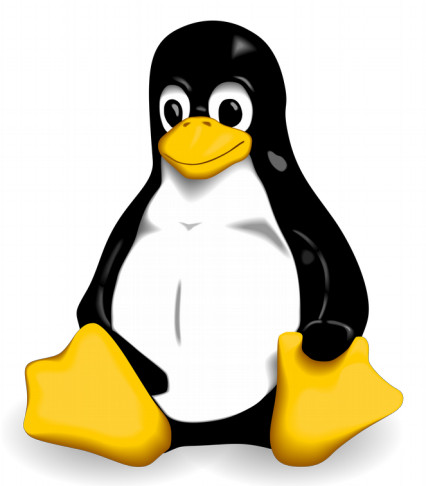
Decryption definition

Decryption is the process of decoding encrypted information so that is can be accessed again by authorized users.

Encryption – Decryption Cycle

To make the data confidential, data(plain text) is encrypted using a particular algorithm and a secret key. After encryption process, plain text gets converted into cipher text. To decrypt the cipher text, similar algorithm is used and at the end the original data is obtained again.

OPERATING SYSTEM



2.1. The Linux operating system

In March 1991 Linus Benedict Torvalds bought the multitasking system Minix for his AT 386. He used it to develop his own multitasking system which he called Linux. In September 1991 he released the first prototype by e-mail to some other Minix users on the internet, thus beginning the Linux project. Many programmers from that point on have supported Linux. They have added device drivers, developed applications, and aimed for POSIX compliance. Today Linux is very powerful, but what is best is that it’s free. Work is beeing done to port Linux to other platforms.

Basically, the application has been written for Linux operating systems which can support ANSI C, C99 etc standards.

The Linux open source operating system, or Linux OS, is a freely distributable, cross-platform operating system based on Unix that can be installed on PCs, laptops, netbooks, mobile and tablet devices, video game consoles, servers, supercomputers and more.

The Linux OS is frequently packaged as a Linux distribution for both desktop and server use, and includes the Linux kernel (the core of the operating system) as well as supporting tools and libraries. Popular Linux OS distributions include Debian, Ubuntu, Fedora, Red Hat and openSUSE.

What is Linux

Linux is, in simplest terms, an operating system. It is the software on a computer that enables applications and the computer operator to access the devices on the computer to perform desired functions. The operating system (OS) relays instructions from an application to, for instance, the computer's processor. The processor performs the instructed task, then sends the results back to the application via the operating system.

Explained in these terms, Linux is very similar to other operating systems, such as Windows and OS X.

But something sets Linux apart from these operating systems. The Linux operating system represented a $25 billion ecosystem in 2008. Since its inception in 1991, Linux has grown to become a force in computing, powering everything from the New York Stock Exchange to mobile phones to supercomputers to consumer devices.

As an open operating system, Linux is developed collaboratively, meaning no one company is solely responsible for its development or ongoing support. Companies participating in the Linux economy share research and development costs with their partners and competitors. This spreading of development burden amongst individuals and companies has resulted in a large and efficient ecosystem and unheralded software innovation.

Over 1,000 developers, from at least 100 different companies, contribute to every kernel release. In the past two years alone, over 3,200 developers from 200 companies have contributed to the kernel--which is just one small piece of a Linux distribution.

This article will explore the various components of the Linux operating system, how they are created and work together, the communities of Linux, and Linux's incredible impact on the IT ecosystem.

Where is Linux?

Linux is already successful on many different kinds of devices, but there are also many technological areas where Linux is moving towards, even as desktop and server development continues to grow faster than any other operating system today.

Linux is being installed on the system BIOS of laptop and notebook computers, which will enable users to turn their devices on in a matter of seconds, bringing up a streamlined Linux environment. This environment will have Internet connectivity tools such as a web browser and an e-mail client, allowing users to work on the Internet without having to boot all the way into their device's primary operating system--even if that operating system is Windows.

At the same time, Linux is showing up on mobile Internet devices (MIDs). This includes embedded devices such as smartphones and PDAs, as well as netbook devices--small laptop-type machines that feature the core functionality of their larger counterparts in a smaller, more energy-efficient package.

The growth of cloud computing is a natural fit for Linux, which already runs many of the Internet's web servers. Linux enables cloud services such as Amazon's A3 to work with superior capability to deliver online applications and information to users.

Related to Linux' growth in cloud computing is the well-known success of Linux on supercomputers, both in the high-performance computing (HPC) and high-availability (HA) areas, where academic research in physics and bioengineering, and firms in the financial and energy industries need reliable and scalable computing power to accomplish their goals.

Many of the popular Web 2.0 services on the Internet, such as Twitter, Linked In, YouTube, and Google all rely on Linux as their operating system. As new web services arrive in the future, Linux will increasingly be the platform that drives these new technologies.

The Birth of Linux

On August 25, 1991, a Finn computer science student named Linus Torvalds made the following announcement to the Usenet group comp.os.minux:

"I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones. This has been brewing since april, and is starting to get ready. I'd like any feedback on things people like/dislike in minix, as my OS resembles it somewhat (same physical layout of the file-system (due to practical reasons) (among other things)."

The “Minix” Torvalds referred to is a variant of the UNIX operating system, used as a guideline for his the free operating system he wanted to run on the x86-based consumer PCs of the day. “gnu” refers to the set of GNU (GNU Is Not Unix) tools first put together by Richard Stallman in 1983. UNIX, the operating system that started it all, had its origins in the old Bell Labs back in the early 60s.

Torvalds built the core of the Linux operating system, known as the kernel. A kernel alone does not

make an operating system, but Stallman's GNU tools were from a project to create an operating system as well--a project that was missing a kernel to make Stallman's operating system complete. Torvalds' matching of GNU tools with the Linux kernel marked the beginning of the Linux operating system as it is known today.

Linux is in many ways still only at the beginning of its potential, even though it has enjoyed tremendous success since Torvalds' first request for help in 1991.

Linux has gained strong popularity amongst UNIX developers, who like it for its portability to many platforms, its similarity to UNIX, and its free software license. Around the turn of the century, several commercial developers began to distribute Linux, including VA Linux, TurboLinux, Mandrakelinux, Red Hat, and SuSE GMbH. IBM's 2000 decision to invest $2 billion in Linux development and sales was a significant positive event to the growth of Linux.

Today, Linux is a multi-billion dollar industry, with companies and governments around the world taking advantage of the operating system's security and flexibility. Thousands of companies use Linux for day-to-day use, attracted by the lower licensing and support costs. Governments around the world are deploying Linux to save money and time, with some governments commissioning their own versions of Linux.

The analyst group IDC has projected Linux will be a $49 billion business by 2011, and there are many indications in the market that this figure will be achieved.

The Code

Linux is also unique from other operating systems in that it has no single owner. Torvalds still manages the development of the Linux kernel, but commercial and private developers contribute other software to make the whole Linux operating system.

In this section, the parts of the Linux operating system will be examined.

2.2. The Kernel

All operating systems have kernels, built around the architectural metaphor that there must be a central set of instructions to direct device hardware, surrounded by various modular layers of functionality. The Linux kernel is unique and flexible because it is also modular in nature.

The kernel of the Window operating system (which few people outside of Microsoft are allowed to look at without paying for the privilege) is a solidly connected piece of code, unable to be easily broken up into pieces. It is difficult (if not impossible) to pare down the Windows kernel to fit on a phone.

This modularity is significant to the success of Linux. The ability to scale down (or up) to meet the needs of a specific platform is a big advantage over other operating systems constrained to just a few possible platforms.

Modularity also effects stability and security as well. If one piece of the kernel code happens to fail, the rest of the kernel will not crash. Similarly, an illicit attack on one part of the kernel (or the rest of the operating system) might hamper that part of the code, but should not compromise the security of the whole device.

“The base of Linux is the kernel. You could replace each and every library, but as long as the Linux kernel remained, it would still be Linux. The kernel contains device drivers, memory management, process management and communication management. The kernel hacker gurus follow POSIX guidelines which sometimes makes programming easier and sometimes harder. If your program behaves differently on a new Linux kernel release, chances are that a new POSIX guideline has been implemented. For programming information about the Linux kernel, read the Linux Kernel Hacker’s Guide.”

2.3. The Operating System

Developers need special tools (like the compilers and command lines found in GNU) to write applications that can talk to the kernel. They also need tools and applications to make it easy for outside applications to access the kernel after the application is written and installed.

This collective set of tools, combined with a kernel, is known as the operating system. It is generally the lowest layer of the computer's software that is accessible by the average user. General users get to the operating system when they access the command line.

Linux provides powerful tools with which to write their applications: developer environments, editors, and compilers are designed to take a developer's code and convert it to something that can access the kernel and get tasks done.

Like the kernel, the Linux operating system is also modular. Developers can pick and choose the operating tools to provide users and developers with a new flavor of Linux designed to meet specific tasks.

The Environments

The windows, menus, and dialog boxes most people think of as part of the operating system are actually separate layers, known as the windowing system and the desktop environment.

These layers provide the human-oriented graphical user interface (GUI) that enables users to easily work with applications in the operating system and third-party applications to be installed on the operating system.

In Linux, there a lot of choices for which windowing system and desktop environment can be used, something that Linux allows users to decide. This cannot be done in Windows and it's difficult to do in OS X.

Like the operating system and kernel, there are tools and code libraries available that let application

developers to more readily work with these environments (e.g., gtk+ for GNOME, Qt for KDE).

2.4. The Applications

Operating systems have two kinds of applications: those that are essential components of the operating system itself, and those that users will install later. Closed operating systems, like Windows and OS X, will not let users (or developers) pick and choose the essential component applications they can use. Windows developers must use Microsoft's compiler, windowing system, and so on.

Linux application developers have a larger set of choices to develop their application. This allows more flexibility to build an application, but it does mean a developer will need to decide which Linux components to use.

The Distributions

This is the highest layer of the Linux operating system: the container for all of the aforementioned layers. A distribution's makers have decided which kernel, operating system tools, environments, and applications to include and ship to users.

Distributions are maintained by private individuals and commercial entities. A distribution can be installed using a CD that contains distribution-specific software for initial system installation and configuration. For the users, most popular distributions offer mature application management systems that allow users to search, find, and install new applications with just a few clicks of the mouse.

There are, at last count, over 350 distinct distributions of Linux.

Licensing

Code is contributed to the Linux kernel under a number of licenses, but all code must be compatible with version 2 of the GNU General Public License (GPLv2), which is the license covering the kernel distribution as a whole. In practice, that means that all code contributions are covered either by GPLv2 (with, optionally, language allowing distribution under later versions of the GPL) or the three-clause BSD license. Any contributions which are not covered by a compatible license will not be accepted into the kernel.

Copyright assignments are not required (or requested) for code contributed to the kernel. All code merged into the mainline kernel retains its original ownership; as a result, the kernel now has thousands of owners.

One implication of this ownership structure is that any attempt to change the licensing of the kernel is doomed to almost certain failure. There are few practical scenarios where the agreement of all copyright holders could be obtained (or their code removed from the kernel). So, in particular, there is no prospect of a migration to version 3 of the GPL in the foreseeable future.

It is imperative that all code contributed to the kernel be legitimately free software. For that reason,

code from anonymous (or pseudonymous) contributors will not be accepted. All contributors are required to "sign off" on their code, stating that the code can be distributed with the kernel under the GPL. Code which has not been licensed as free software by its owner, or which risks creating copyright-related problems for the kernel (such as code which derives from reverse-engineering efforts lacking proper safeguards) cannot be contributed.

Questions about copyright-related issues are common on Linux development mailing lists. Such questions will normally receive no shortage of answers, but one should bear in mind that the people answering those questions are not lawyers and cannot provide legal advice. If you have legal questions relating to Linux source code, there is no substitute for talking with a lawyer who understands this field. Relying on answers obtained on technical mailing lists is a risky affair.

Community

Linux communities come in two basic forms: developer and user communities.

One of the most compelling features of Linux is that it is accessible to developers; anybody with the requisite skills can improve Linux and influence the direction of its development. Proprietary products cannot offer this kind of openness, which is a characteristic of the free software process.

Developer communities can volunteer to maintain and support whole distributions, such as the Debian or Gentoo Projects. Novell and Red hat also support community-driven versions of their products, openSUSE and Fedora, respectively. The improvements to these community distros are then incorporated into the commercial server and desktop products from these companies.

The Linux kernel itself is primarily supported by its developer community as well and is one of the largest and most active free software projects in existence. A typical three-month kernel development cycle can involve over 1000 developers working for moreThat's still the same way:

(1) Place the cursor in the header or footer

(2) Menu Insert > Fields > Page number

than 100 different companies (or for no company at all).

With the growth of Linux has come an increase in the number of developers (and companies) wishing to participate in its development. Hardware vendors want to ensure that Linux supports their products well, making those products attractive to Linux users. Embedded systems vendors, who use Linux as a component in an integrated product, want Linux to be as capable and well-suited to the task at hand as possible. Distributors and other software vendors who base their products on Linux have a clear interest in the capabilities, performance, and reliability of the Linux kernel.

Other developer communities focus on different applications and environments that run on Linux, such as Firefox, OpenOffice.org, GNOME, and KDE.

End users, too, can make valuable contributions to the development of Linux. With online communities such as Linux.com, LinuxQuestions, and the many and varied communities hosted by distributions and applications, the Linux user base is an often vocal, usually positive advocate and guide for the Linux operating system.

The Linux community is not just a presence online. Local groups known as Linux Users Groups (LUGs) often meet to discuss issues regarding the Linux operating system, and provide other local users with free demonstrations, training, technical support, and installfests.

2.5. The Linux libc package

libc: ISO 8859.1, < linux/param.h >, YP functions, crypt functions, some basic shadow routines (by default not included), … old routines for compatibility in libcompat (by default not activated), english, french or german error messages, bsd 4.4lite compatible screen handling routines in libcurses, bsd compatible routines in libbsd, screen handling routines in libtermcap, database management routines in libdbm, mathematic routines in libm, entry to execute programs in crt0.o ???, byte sex information in libieee ??? (could someone give some infos instead of laughing ?), user space profiling in libgmon. I wish someone of the Linux libc developers would write this chapter. All i can say now that there is going to be a change from the a.out executable format to the elf (executable and linkable format) which also means a change in building shared libraries. Currently both formats (a.out and elf) are supported.

2.6. System calls

A system call is usually a request to the operating system (kernel) to do a hardware/system- specific or privileged operation. As of Linux-1.2, 140 system calls have been defined. System calls like close() are implemented in the Linux libc. This implementation often involves calling a macro which eventually calls syscall(). Parameters passed to syscall() are the number of the system call followed by the needed arguments. The actual system call numbers can be found in < linux/unistd.h > while < sys/syscall.h > gets updated with a new libc. If new calls appear that don't have a stub in libc yet, you can use syscall(). As an example, you can close a file using syscall() like this (not advised):

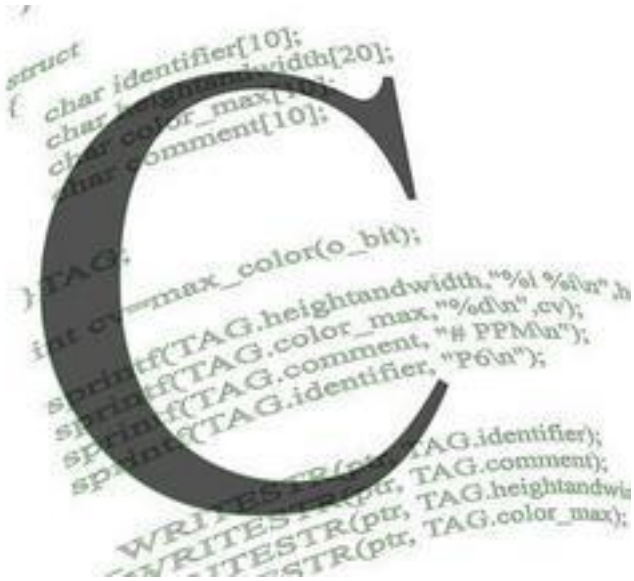
```
#include <syscall.h>
extern int syscall(int, ...);
int my_close(int filedescriptor)
{
return syscall(SYS_close, filedescriptor);
}
```

On the i386 architecture, system calls are limited to 5 arguments besides the system call number because of the number of hardware registers. If you use Linux on another architecture you can check < asm/unistd.h > for the syscall macros to see how many arguments your hardware supports or how many the developers chose to support. These syscall macros can be used instead of syscall(), but this is not recommended since such a macro expands to a full function which might already exist in a library. Therefore, only kernel hackers should play with the syscall macros. To demonstrate, here is the close() example using a syscall macro.

```
#include <linux/unistd.h>
_syscall1(int, close, int, filedescriptor);
```

The syscall1 macro expands revealing the close() function. Thus we have close() twice–once in libc and once in our program. The return value of syscall() or a syscall macro is -1 if the system call failed and 0 or greater on success. Take a look at the global variable errno to see what happened if a system call failed. The following system calls that are available on BSD and SYS V are not available on Linux:
audit(), auditon(), auditsvc(), fchroot(), getauid(), getdents(), getmsg(), mincore(), poll(), putmsg(), setaudit(), setauid().

PROGRAMMING LANGUAGE



C PROGRAMMING

C is a general purpose, block structured, midlevel programming language. It was initially developed for writting UNIX operating system, but now it is a general purpose programming language. It is used to develope various system softwares and application softwares. Even compilers of many programming languages is also written in C language. C was developed by Deniss Ritchie and Brian Kernighan at AT & T Bell Laboratories in 1972. It is the combination of two BCPL & B languages. It came after the language B, so It was named C. It is a procedural programming language i.e., any problem is devided into various functions and program is executed from top to buttom. Data is passed from function to function and every function changes its forms and at last the output is given. It follows top-buttom procedure procedure in order to execute a program. C language can be used to write the system software as well as application software, it means, it can be used as high level languge as well as low level language, that's why it is called midlevel laguage.

There are many standards of C language. Inititally it was given the D&K standard after the name of Ritchie and Kernighan. After that there were many standards given. ANSI C was come in existence in 1980s. The most popular standards are C99, C89, ANSI C.

The first high level language was BASIC(Beginners All Purpose Symbolic Instruction Code) developed by Microsoft for designing application softwares. After that many standards were given and many languages were come in existence but their usage were limited. Only computer experts could work on those. But, as soon as C come in existence, it became popular.

Learning C language is very analogous to learn our natural language. As we learn first of all the character set of a natural language, C has also its character set. Word is the meaningful combination of the characters, in C these are called identifiers. A meaningful collection of word is phrase, in C word is analogous to a expression. A set of phrases is called a sentence, in C sentence is called statement. The set of sentences creates a paragraph, and set of statements is called program. The set of programs is called software. Every language has its own grammer which must be followed to write anything in that language, so programming languages have also their own grammers for writting softwares.

Meaning to say high level language is that the code of the programs is written like english language(symbolic codes). We have three types of machine languages :-

1. Binary Codes :-

It is the language in which machine executes any instructions. It contains only '1s' and '0s'. '1' means the high logic level and '0' means the low logic level. For example if a bulb glows only for more than 6V and below 6V it cant glow, then for that bulb, above 6V will be the high logic level and below that it is low logic level. Same on-off state is used in the microprocessors. For on-off switching transistors are used. It is a machine dependent language, i.e., every processor has its own syntax and form of the codes.

2. Assembly Languages :-

Assembly languages are also machine dependent languages. In this language, first of all we write the code in the form of set of instructions, which can be understood by human. After that it will be converted into the hexadecimal codes written in the manual of the microprosessor,s manual. The

hexadecimal codes is then converted into the binary codes by a set of programs called monitor program. Monitor program is stored in the RAM of the microprocessor kit. It is also called assembler.

3. High Level Languages:-

High level languages are also called the symbolic instruction code, which can be understood by human. It is english like language. The first high level language was BASIC. Since any type of microprocessor can only operate on the binary codes. So, any high level language need to be converted into the binary codes before execution. The set of codes which convert the high level language into the binary codes is called the compiler. Compiler converts the symbolic instructions into the assembly language and then binary codes.

A brief introduction of C programming language in Linux is given below :-

3.1. Character Set of C:-

Every programming language has its own set of characters which are used to write the statements of the program (source code). In C language also, a program is written by using a set of characters. These characters are known as alphabet or character set of C. The alphabet of C language is given below.

1. Lowercase letters: a, b, c, ..., z
2. Uppercase letters: A, B, C, ..., Z
3. Digits: 0, 1, 2, ..., 9
4. Special characters: +, -, /, =, (,), [,], {, }, <, >, ', ", !, @, #, $, %, \, /, ;
5. White Spaces: blanks, new line, tab etc.

3.2. Identifier :

Identifiers are the name of any variable name like macro name, function name, macro definition, macro parameter, union, structure etc. There are some rules to write an identifiers:

1. All the upper case, lower case characters, digits can be used to create an identifier.

2. First character of any identifier cannot be a digit.

3. Any type of special characters even blank space are not allowed in the identifier except under-score, which can be used inside the identifiers.

4. At any situation, keywords can not be a valid identifiers.

Identifiers are chosen such that we can determine its work only by its name.

3.3. Keywords :

Keywords are some reserved words which are used for specific purposes and these can not be used as any identifier name.

Standard ANSI C recognizes the following keywords:

auto, break, case, char, const, countinue, default, do, double, else, enum, extern, float, for, goto, if, int, long, register, return, short, signed, sizeof, static, struct, switch, typedef, union, unsigned, void, volatile, while.

3.4. Declaration Statement :

An identifier can be declared by making use of the declaration statements. The role of a declaration statement is to introduce the name of an identifier along with its data type to the compiler before its use. The general form of a declaration statement is :

[storage-class-specifier][type_qualifier][type_modifier]type_identifier[=value[…]];

3.5. Data Types :

Data types is one of the most important attributes of an identifier. It determines the possible values that an identifier can have and the valid operations that can be applied on it.

In C language data types are broadly classified as :-

1. Basic data type (Primitive Data Type)

a. Character (char)

b. integer (int)

c. Single precision floating point (float)

d. Double precision floating point (double)

e. no value available (void)

2. Derived Data Types

a. array type

b. Pointer type

c. Function Type

3. User Defined Data Types

a. Structure

b. Union

c. Enumeration

3.6. Type Qualifier and Type Modifier

A **type qualifier** neither effects the range of values nor the arithmetic properties of the declared object. They are used to indicate the special properties of data within an object. Two type qualifiers available in C :

const Qualifier : Declaring an object const announces that its value will not be changed during the execution of a program.

Volatile Qualifier : It announces that the object has some special properties relevant to optimization.

A **type modifier** modifies the base type to yield a new type. It modifies the range and the arithmetic properties of the base type. The type modifiers and the corresponding keywords available in C are :

1. Signed(signed)

2. Unsigned(unsigned)

3. Long (long)

4. Short (short)

3.7. Data Object :

Data object is a term that is used to specify the region of the data storage that is used to hold values. Once an identifier is allocated memory space, It will be known as a data object.

L-value :

L-value is a data object locator. The term l-value can be further categorized as :

1. Modifiable l-value : A modifiable l-value is an expression that refers to an object that can be accessed an legally changed in the memory.

2. Non-modifiable l-value : A non-modifiable l-value refers to an object that can be accesses but cannot be changed in the memory.

R-value :

R-value refers to read value.

Variables :

A variable is an entity whose value can vary during the execution of a program. The value of a variable can be changed because it has a modifiable l-value. Since it has a modifiable l-value, it can be placed

on the left side of the assignment operator. The variable can also be placed on the right of the operator. Hence it has r-value too.

Constant :

A constant is an entity whose value remains the same throughout the execution of a program. It cannot be placed on the left side of the assignment operator because it does have a modifiable l-value. It can only be placed on the right side of the assignment operator. Thus a constant has an r-value only.

There are three types of constants :

1. Literal Constant

2. Symbolic Constant

2. Qualified Constant

Escape Sequences :-

S.No.	Escape Sequence	Character Value
1	\'	Prints '
2	\”	Printd “
3	\?	Prints ?
4	\a	Alert by generating a beep
5	\b	Backspace
6	\f	Form feed
7	\n	New line
8	\r	Carriage return
9	\t	Horizontal tab
10	\v	Vertical tab
11	\0	Null character
12	\\	Backslash character (\)

Format Specifiers

S.No.	Data Type	Format Specifier	Remark
1.	char	%c	Signed character
2.	Int	%i	Signed Integer
3.	Int	%d	Signed Integer in decimal

4.	Unsigned int	%o	unsigned integer in octal
5.	Unsigned int	%x	unsigned Integer in hexadecimal
6.	Long int	%ld	Signed long
7.	Short int	%hd	Signed short
8.	Unsigned long	%lu	Unsigned long
9.	Unsigned short	%hu	Unsigned short
10.	Float	%f	Single precision floating point
11.	Double	%lf	Double precision floating point
12.	String	%s	String type
13	Pointer	%p	Pointer type

STRUCTURE OF A C PROGRAM

1. Preprocessor Directives

2. Global Declaration

3. Functions

3.8. OPERATORS & EXPRESSIONS

1. The symbols which are used to perform logical and mathematical operations in a C program are called C operators.

2.These C operators join individual constants and variables to form expressions.

3. Operators, functions, constants and variables are combined together to form expressions.

4. Consider the expression A + B * 5. where, +, * are operators, A, B are variables, 5 is constant and A + B * 5 is an expression.

Types of C operators:

C language offers many types of operators. They are,

1.Arithmetic operators
2.Assignment operators
3.Relational operators
4.Logical operators
5.Bit wise operators
6.Conditional operators (ternary operators)

7.Increment/decrement operators

8.Special operators

S.no	Types of Operators	Description
1	**Arithmetic_operators**	These are used to perform mathematical calculations like addition, subtraction, multiplication, division and modulus
2	**Assignment_operators**	These are used to assign the values for the variables in C programs.
3	**Relational operators**	These operators are used to compare the value of two variables.
4	**Logical operators**	These operators are used to perform logical operations on the given two variables.
5	**Bit wise operators**	These operators are used to perform bit operations on given two variables.
6	**Conditional (ternary) operators**	Conditional operators return one value if condition is true and returns another value is condition is false.
7	**Increment/decrement operators**	These operators are used to either increase or decrease the value of the variable by one.
8	**Special operators**	&, *, sizeof( ) and ternary operators.

NAME

operator - C operator precedence and order of evaluation

DESCRIPTION

This manual page lists C operators and their precedence in evaluation.

Operator	**Associativity**
() [] -> .	left to right
! ~ ++ -- + - (type) * & sizeof	right to left
* / %	left to right
+ -	left to right
<< >>	left to right
< <= > >=	left to right
== !=	left to right
&	left to right
^	left to right
|	left to right

&&	left to right
||	left to right
?:	right to left
= += -= *= /= %= <<= >>= &= ^= |=	right to left
,	Left to right

3.9. Decision Control Statements

In decision control statements (C if else and nested if), group of statements are executed when condition is true. If condition is false, then else part statements are executed.

There are 3 types of decision making control statements in C language. They are,

1. if statements
2. if else statements
3. nested if statements

“If”, “else” and “nested if” decision control statements in C:

Syntax for each C decision control statements are given in below table with description.

Decision control statements	**Syntax**	**Description**
if	if (condition) {Statements;}	In these type of statements, if condition is true, then respective block of code is executed.
if…else	if (condition) {Statement1; Statement2; } else {Statement3; Statement4; }	In these type of statements, group of statements are executed when condition is true. If condition is false, then else part statements are executed.
nested if	if (condition1){Statement1;} else_if(condition2) {Statement2;} else Statement 3;	If condition 1 is false, then condition 2 is checked and statements are executed if it is true.If condition 2 also gets failure, then else part is executed.

3.10. LOOPS

Types of loop control statements in C:

Loop control statements in C are used to perform looping operations until the given condition is true. Control comes out of the loop statements once condition becomes false.

There are 3 types of loop control statements in C language. They are,

1.for
2.while
3.do-while

Syntax for each C loop control statements are given in below table with description:

Loops are used to repeat a block of code. Being able to have your program repeatedly execute a block of code is one of the most basic but useful tasks in programming -- many programs or websites that produce extremely complex output (such as a message board) are really only executing a single task many times. (They may be executing a small number of tasks, but in principle, to produce a list of messages only requires repeating the operation of reading in some data and displaying it.) Now, think about what this means: a loop lets you write a very simple statement to produce a significantly greater result simply by repetition.

FOR -

for loops are the most useful type. The syntax for a for loop is

```
 for ( variable initialization; condition; variable update ) {
 Code to execute while the condition is true
}
```

The variable initialization allows you to either declare a variable and give it a value or give a value to an already existing variable. Second, the condition tells the program that while the conditional expression is true the loop should continue to repeat itself. The variable update section is the easiest way for a for loop to handle changing of the variable. It is possible to do things like x++, x = x + 10, or even x = random (5), and if you really wanted to, you could

call other functions that do nothing to the variable but still have a useful effect on the code. Notice that a semicolon separates each of these sections, that is important. Also note that every single one of the sections may be empty, though the semicolons still have to be there. If the condition is empty, it is evaluated as true and the loop will repeat until something else stops it.

This was another simple example, but it is longer than the above FOR loop. The easiest way to think of the loop is that when it reaches the brace at the end it jumps back up to the beginning of the loop, which checks the condition again and decides whether to repeat the block another time, or stop and move to the next statement after the block.

DO..WHILE -

DO..WHILE loops are useful for things that want to loop at least once. The structure is

```
do {

} while ( condition );
```

Notice that the condition is tested at the end of the block instead of the beginning, so the block will be executed at least once. If the condition is true, we jump back to the beginning of the block and execute it again. A do..while loop is almost the same as a while loop except that the loop body is guaranteed to execute at least once. A while loop says "Loop while the condition is true, and execute this block of code", a do..while loop says "Execute this block of code, and then continue to loop while the condition is true".

Break and Continue

Two keywords that are very important to looping are break and continue. The break command will exit the most immediately surrounding loop regardless of what the conditions of the loop are. Break is useful if we want to exit a loop under special circumstances. For example, let's say the program we're working on is a two-person checkers game. The basic structure of the program might look like this:

```
while (true)
{
    take_turn(player1);
    take_turn(player2);
}
```

This will make the game alternate between having player 1 and player 2 take turns. The only problem with this logic is that there's no way to exit the game; the loop will run forever!

3.11. The Case Control In C :

The statements which are used to execute only specific block of statements in a series of blocks are called case control statements.

There are 4 types of case control statements in C language. They are,

1.switch
2.break
3.continue
4.goto

1. switch case statement in C:

•Switch case statements are used to execute only specific case statements based on the switch expression.
•Below is the syntax for switch case statement.

switch (expression)
{
case label1: statements;
break;
case label2: statements;
break;
default: statements;
break;
}

2. break statement in C:

Break statement is used to terminate the while loops, switch case loops and for loops from the subsequent execution.

- Syntax: break;

3. Continue statement in C:

Continue statement is used to continue the next iteration of for loop, while loop and do-while loops. So, the remaining statements are skipped within the loop for that particular iteration.

Syntax : continue;

4. goto statement in C:

- goto statements is used to transfer the normal flow of a program to the specified label in the program.
- Below is the syntax for goto statement in C.

```
{
…….
go to label;
…….
…….
LABEL:
statements;
}
```

Storage class specifiers in C language tells the compiler where to store a variable, how to store the variable, what is the initial value of the variable and life time of the variable.

Syntax:

storage_specifier data_type variable _name

3.12. STORAGE CLASS SPECIFIERS

Types of Storage Class Specifiers in C:

There are 4 storage class specifiers available in C language. They are,

1.auto
2.extern
3.static
4.register

S. No.	Storage Specifier	Storage place	Initial / default value
1	**auto**	CPU Memory	Garbage value
2	**extern**	CPU memory	Zero
3	**static**	CPU memory	Zero
4	**register**	Register memory	Garbage value

Note:

•For faster access of a variable, it is better to go for register specifiers rather than auto specifiers.

•Because, register variables are stored in register memory whereas auto variables are stored in main CPU memory.

•Only few variables can be stored in register memory. So, we can use variables as register that are used very often in a C program.

3.13. ARRAY:

C Array is a collection of variables belongings to the same data type. You can store group of data of same data type in an array.

- Array might be belonging to any of the data types
- Array size must be a constant value.
- Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
- It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

Example for C Arrays:

- int a[10]; // integer array
- char b[10]; // character array i.e. string

Types of C arrays:

There are 2 types of C arrays. They are,

1. One dimensional array
2. Multi dimensional array
    1. Two dimensional array
    2. Three dimensional array, four dimensional array etc…

1. One dimensional array in C:

- Syntax : data-type arr_name[array_size];

Array declaration	Array initialization	Accessing array
Syntax: data_type arr_name [arr_size];	data_type arr_name [arr_size]=(value1, value2, value3,….);	arr_name[index];
int age [5];	int age[5]={0, 1, 2, 3, 4};	age[0];_/*0_is_accessed*/age[1];_/*1_is_accessed*/age[2];_/*2_is_accessed*/
char str[10];	Char str[10]={'H','a','i'}; (or)char str[0] = 'H';char str[1] = 'a'; char str[2] = 'i;	str[0];_/*H is accessed*/str[1]; /*a is accessed*/str[2]; /* i is accessed*/

2. Two dimensional array in C:

- Two dimensional array is nothing but array of array.
- syntax : data_type array_name[num_of_rows][num_of_column]

S.no	Array declaration	Array initialization	Accessing array
1	Syntax: data_type arr_name [num_of_rows] [num_of_column];	data_type arr_name[2][2] = {{0,0},{0,1},{1,0},{1,1}};	arr_name[index];
2	Example:int arr[2][2];	int arr[2][2] = {1,2, 3, 4};	Arr [0] [0] = 1; arr [0]]1] = 2;arr [1][0] arr [1] [1] = 4;

3.14. STRING

C Strings are nothing but array of characters ended with null character ('\0').

- This null character indicates the end of the string.
- Strings are always enclosed by double quotes. Whereas, character is enclosed by single quotes in C.

C String functions:

- String.h header file supports all the string functions in C language. All the string functions are given below.
- Click on each string function name below for detail description and example programs.

S.no	String functions	Description
1	**strcat ( )**	Concatenates str2 at the end of str1.
2	**strncat ( )**	appends a portion of string to another
3	**strcpy ( )**	Copies str2 into str1
4	**strncpy ( )**	copies given number of characters of one string to another
5	**strlen ( )**	gives the length of str1.
6	**strcmp ( )**	Returns 0 if str1 is same as str2. Returns <0 if strl <

		str2. Returns >0 if str1 > str2.
7	**strcmpi_(.)**	Same as strcmp() function. But, this function negotiates case. “A” and “a” are treated as same.
8	**strchr ( )**	Returns pointer to first occurrence of char in str1.
9	**strrchr ( )**	last occurrence of given character in a string is found
10	**strstr ( )**	Returns pointer to first occurrence of str2 in str1.
11	**strrstr ( )**	Returns pointer to last occurrence of str2 in str1.
12	**strdup ( )**	duplicates the string
13	**strlwr ( )**	converts string to lowercase
14	**strupr ( )**	converts string to uppercase
15	**strrev ( )**	reverses the given string
16	**strset ( )**	sets all character in a string to given character
17	**strnset ( )**	It sets the portion of characters in a string to given character
18	**strtok ( )**	tokenizing given string using delimiter

3.15. POINTERS

- C Pointer is a variable that stores/points the address of another variable. C Pointer is used to allocate memory dynamically i.e. at run time. The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.

Syntax : data_type *var_name; Example : int *p; char *p;

• Where, * is used to denote that “p” is pointer variable and not a normal variable.

• **Key points to remember about pointers in C:**

1. Normal variable stores the value whereas pointer variable stores the address of the variable.

2.The content of the C pointer always be a whole number i.e. address.

3. Always C pointer is initialized to null, i.e. int *p = null.

4. The value of null pointer is 0.

5. & symbol is used to get the address of the variable.

6. * symbol is used to get the value of the variable that the pointer is pointing to.

7. If pointer is assigned to NULL, it means it is pointing to nothing.

8.Two pointers can be subtracted to know how many elements are available between these two pointers.

3.16. FUNCTIONS

1. What is C function?

A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

2. Uses of C functions:

- C functions are used to avoid rewriting same logic/code again and again in a program.
- There is no limit in calling C functions to make use of same functionality wherever required.
- We can call functions any number of times in a program and from any place in a program.
- A large C program can easily be tracked when it is divided into functions.
- The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

3. C function declaration, function call and function definition:

There are 3 aspects in each C function. They are,

- Function declaration or prototype - This informs compiler about the function name, function parameters and return value's data type.
- Function call – This calls the actual function
- Function definition – This contains all the statements to be executed.

S.no	C function aspects	syntax
1	function definition	return_type function_name (arguments list) { Body of function; }
2	function call	function_name (arguments list);
3	function declaration	return_type function_name (argument list);

4. How to call C functions in a program?

There are two ways that a C function can be called from a program. They are,

1.Call by value
2.Call by reference

1. Call by value:

- In call by value method, the value of the variable is passed to the function as parameter.
- The value of the actual parameter can not be modified by formal parameter.
- Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

Note:

- Actual parameter – This is the argument which is used in function call.
- Formal parameter – This is the argument which is used in function definition

2. Call by reference:

- In call by reference method, the address of the variable is passed to the function as parameter.
- The value of the actual parameter can be modified by formal parameter.
- Same memory is used for both actual and formal parameters since only address is used by both parameters.

C – Argument, return value

All C functions can be called either with arguments or without arguments in a C program. These functions may or may not return values to the calling function. Now, we will see simple example C programs for each one of the below.

1.C function with arguments (parameters) and with return value
2.C function with arguments (parameters) and without return value
3.C function without arguments (parameters) and without return value
4.C function without arguments (parameters) and with return value

S.no	C function	syntax

1	with arguments and with return values	int function (int); // function declaration
		function (a); // function call
		int function(int a) // function definition
		{statements; return a;}

2	with arguments and without return values	void function (int); // function declaration
		function(a); // function call
		void function(int a) // function definition
		{statements;}

3	without arguments and without return values	void function(); // function declaration
		function(); // function call
		void function() // function definition
		{statements;}

4	without arguments and with return values	int function ( ); // function declaration
		function ( ); // function call
		int function( ) // function definition
		{statements; return a;}

Note:

•If the return data type of a function is “void”, then, it can’t return any values to the calling function.

•If the return data type of the function is other than void such as “int, float, double etc”, then, it can return values to the calling function.

C – Library functions

•Library functions in C language are inbuilt functions which are grouped together and placed in a common place called library.

•Each library function in C performs specific operation.

•We can make use of these library functions to get the pre-defined output instead of

writing our own code to get those outputs.

•These library functions are created by the persons who designed and created C compilers.

•All C standard library functions are declared in many header files which are saved as file_name.h.

•Actually, function declaration, definition for macros are given in all header files.

•We are including these header files in our C program using "#include<file_name.h>" command to make use of the functions those are declared in the header files.

•When we include header files in our C program using "#include<filename.h>" command, all C code of the header files are included in C program. Then, this C program is compiled by compiler and executed.

Command line arguments in C:

main() function of a C program accepts arguments from command line or from other shell scripts by following commands. They are,

•argc

•argv[]

where,

argc - Number of arguments in the command line including program name argv[] – This is carrying all the arguments

• In real time application, it will happen to pass arguments to the main program itself. These arguments are passed to the main () function while executing binary file from command line.

• For example, when we compile a program (test.c), we get executable file in the name "test".

• Now, we run the executable "test" along with 4 arguments in command line like below.

Example program for argc() and argv() functions in C:

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])   //  command line arguments
```

```
{
if(argc!=5)
{
   printf("Arguments passed through command line " \
          "not equal to 5");
   return 1;
}

   printf("\n Program name  : %s \n", argv[0]);
   printf("1st arg  : %s \n", argv[1]);
   printf("2nd arg  : %s \n", argv[2]);
   printf("3rd arg  : %s \n", argv[3]);
   printf("4th arg  : %s \n", argv[4]);
   printf("5th arg  : %s \n", argv[5]);

return 0;
}
```

Output:

```
Program name : test
1st arg : this
2nd arg : is
3rd arg : a
4th arg : program
5th arg : (null)
```

C – Variable length argument

- Variable length arguments is an advanced concept in C language offered by c99 standard. In c89 standard, fixed arguments only can be passed to the functions.
- When a function gets number of arguments that changes at run time, we can go for variable length arguments.
- It is denoted as … (3 dots)
- stdarg.h header file should be included to make use of variable length argument functions.

3.17. Dynamic memory allocation in C:

The process of allocating memory during program execution is called dynamic memory allocation.

Dynamic memory allocation functions in C:

C language offers 4 dynamic memory allocation functions. They are,

1.malloc()
2.calloc()
3.realloc()
4.free()

S.no	Function	Syntax
1	malloc ()	malloc (number *sizeof(int));
2	calloc ()	calloc (number, sizeof(int));
3	realloc ()	realloc (pointer_name, number * sizeof(int));
4	free ()	free (pointer_name);

1. malloc() function in C:

•malloc () function is used to allocate space in memory during the execution of the program.

•malloc () does not initialize the memory allocated during execution. It carries garbage value.

•malloc () function returns null pointer if it couldn't able to allocate requested amount of memory.

2. calloc() function in C:

•calloc () function is also like malloc () function. But calloc () initializes the allocated memory to zero. But, malloc() doesn't.

3. realloc() function in C:

•realloc () function modifies the allocated memory size by malloc () and calloc () functions to new size.

•If enough space doesn't exist in memory of current block to extend, new block is

allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

4. free() function in C:

•free () function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.

3.18. C – Type Casting functions

Typecasting concept in C language is used to modify a variable from one date type to another data type. New data type should be mentioned before the variable name or value in brackets which to be typecast.

C type casting example program:

- In the below C program, 7/5 alone will produce integer value as 1.
- So, type cast is done before division to retain float value (1.4).

```
#include <stdio.h>
int main ()
{
        float x;
        x = (float) 7/5;
        printf("%f",x);
}
```

Output:

```
1.400000
```

Note:

•It is best practice to convert lower data type to higher data type to avoid data loss.

•Data will be truncated when higher data type is converted to lower. For example, if float is converted to int, data which is present after decimal point will be lost.

Inbuilt typecast functions in C:

•There are many inbuilt typecasting functions available in C language which performs data type conversion from one type to another.

•Click on each function name below for description and example programs.

S.no	Typecast function	Description
1	**atof()**	Converts string to float
2	**atoi()**	Converts string to int
3	**atol()**	Converts string to long
4	**itoa()**	Converts int to string
5	**ltoa()**	Converts long to string

3.19. C – Structure

C Structure is a collection of different data types which are grouped together and each element in a C structure is called member.

1. If you want to access structure members in C, structure variable should be declared.

2.Many structure variables can be declared for same structure and memory will be allocated for each separately.

3. It is a best practice to initialize a structure to null while declaring, if we don't assign any values to structure members.

Difference between C variable, C array and C structure:

- A normal C variable can hold only one data of one data type at a time.
- An array can hold group of data of same data type.
- A structure can hold group of data of different data types
- Data types can be int, char, float, double and long double etc.

Below table explains following concepts in C structure.

1.How to declare a C structure?
2.How to initialize a C structure?
3.How to access the members of a C structure?

Type	Using normal variable	Using pointer variabe

Syntax	struct tag_name
	{
	data type var_name1;
	data type var_name2;
	data type var_name3;
	};
		struct tag_name
		{
		data type var_name1;
		data type var_name2;
		data type var_name3;
		};

Example	struct student
	{
	int mark;
	char name[10];
	float average;
	};
		struct student
		{
		int mark;
		char name[10];
		float average;
		};

Declaring structure variable	struct student report;	struct student *report, rep;
Initializing structure variable	struct student report = {100, “Mani”, 99.5};	struct student rep = {100, “Mani”, 99.5}; report = &rep;
Accessing structure members	report.mark report.name report.average	report -> mark report -> name report -> average

Uses of C structures:

1.C Structures can be used to store huge data. Structures act as a database.

2.C Structures can be used to send data to the printer.

3.C Structures can interact with keyboard and mouse to store the data.

4.C Structures can be used in drawing and floppy formatting.

5.C Structures can be used to clear output screen contents.

6.C Structures can be used to check computer’s memory size etc.

C – Array of Structures

C Structure is collection of different datatypes (variables) which are grouped together. Whereas, array of structures is nothing but collection of structures. This is also called as structure array in C.

A structure can be passed to any function from main function or from any sub function.

•Structure definition will be available within the function only.

•It won’t be available to other functions unless it is passed to those functions by value or by address(reference).

•Else, we have to declare structure variable as global variable. That means, structure variable should be declared outside the main function. So, this structure will be visible to all the functions in a C program.

Passing structure to function in C:

It can be done in below 3 ways.

1.Passing structure to a function by value
2.Passing structure to a function by address(reference)
3.No need to pass a structure – Declare structure variable as global

C – Structure using Pointer

C structure can be accessed in 2 ways in a C program. They are,

1.Using normal structure variable
2.Using pointer variable

Dot(.) operator is used to access the data using normal structure variable and arrow (->) is used to access the data using pointer variable. You have learnt how to access structure data using normal variable in C – Structure topic. So, we are showing here how to access structure data using pointer variable in below C program.

C – Nested Structure

Nested structure in C is nothing but structure within structure. One structure can be declared inside other structure as we declare structure members inside a structure. The structure variables can be a normal structure variable or a pointer variable to access the data. You can learn below concepts in this section.

1.Structure within structure in C using normal variable
2.Structure within structure in C using pointer variable

C – Struct memory allocation

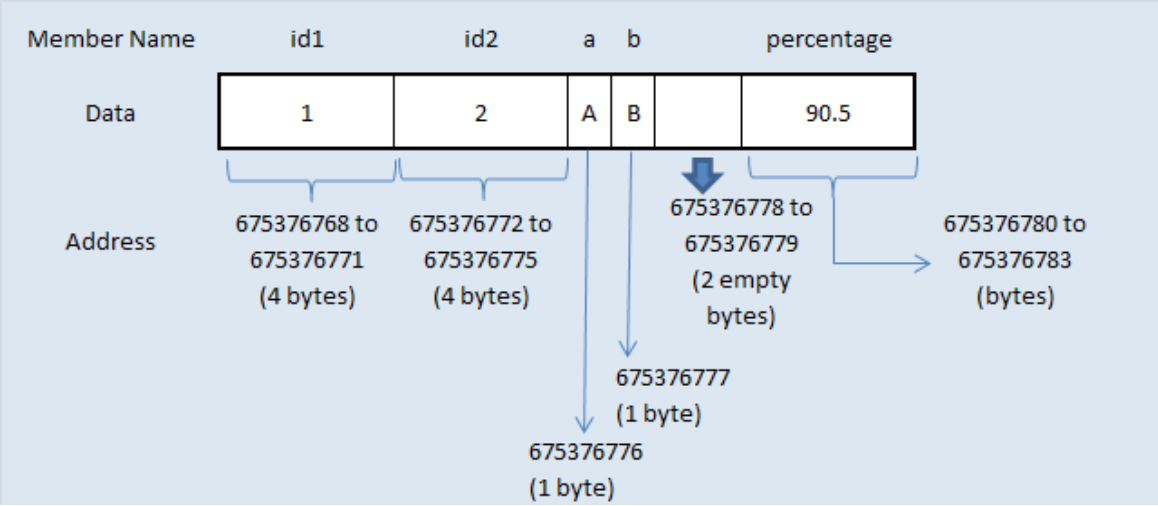
1. How structure members are stored in memory?

Always, contiguous(adjacent) memory locations are used to store structure members in memory. Consider below example to understand how memory is allocated for structures.

Please refer below table to know from where to where memory is allocated for each datatype in contiguous (adjacent) location in memory.

Datatype	Memory allocation in C (32 bit compiler)		
	From Address	To Address	Total bytes
int id1	675376768	675376771	4
int id2	**675376772**	**675376775**	**4**
char a	675376776		1
char b	**675376777**		**1**
Addresses 675376778 and 675376779 are left empty (Do you know why? Please see Structure padding topic below)			**2**
float percentage	675376780	675376783	4

The pictorial representation of above structure memory allocation is given below. This diagram will help you to understand the memory allocation concept in C very easily.



C – Structure Padding

In order to align the data in memory, one or more empty bytes (addresses) are inserted (or left empty) between memory addresses which are allocated for other structure members while memory allocation. This concept is called structure padding.

•Architecture of a computer processor is such a way that it can read 1 word (4 byte in 32 bit processor) from memory at a time.

•To make use of this advantage of processor, data are always aligned as 4 bytes package which leads to insert empty addresses between other member's address.

- Because of this structure padding concept in C, size of the structure is always not same as what we think.

3.20. C – Union

C Union is also like structure, i.e. collection of different data types which are grouped together. Each element in a union is called member.

•Union and structure in C are same in concepts, except allocating memory for their members.
•Structure allocates storage space for all its members separately.
•Whereas, Union allocates one common storage space for all its members

•We can access only one member of union at a time. We can't access all member values at the same time in union. But, structure can access all member values at the same time. This is because, Union allocates one common storage space for all its members. Where as Structure allocates storage space for all its members separately.
•Many union variables can be created in a program and memory will be allocated for each union variable separately.

•Below table will help you how to form a C union, declare a union, initializing and accessing the members of the union.

•Syntax

•union tag_name
{
data type var_name1;
data type var_name2;
data type var_name3;
};

C Preprocessor directives:

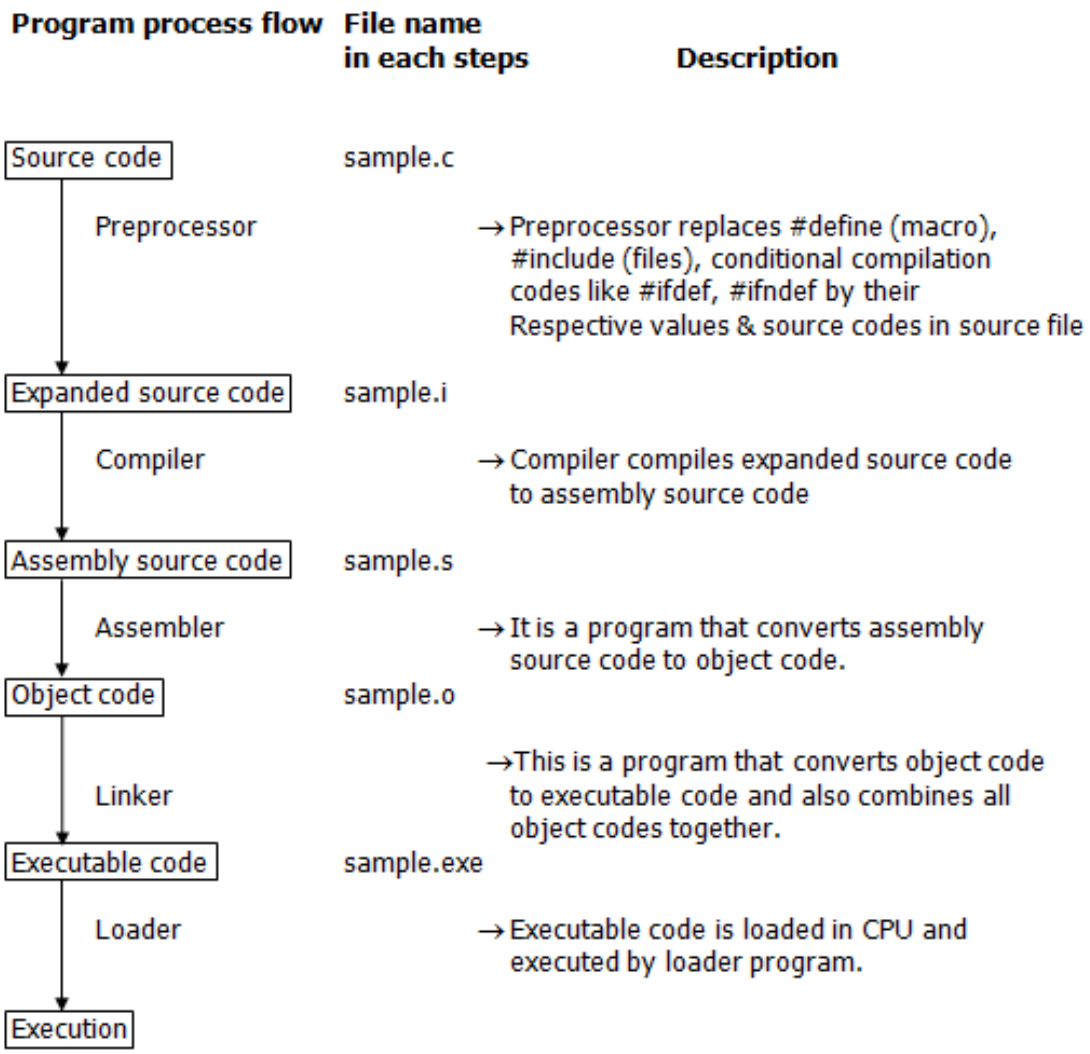
•Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing.

•Commands used in preprocessor are called preprocessor directives and they begin with “#” symbol.

•Below is the list of preprocessor directives that C language offers.

S.no	Preprocessor
1	Macro
2	Header file inclusion
3	Conditional compilation
4	Other directives

A program in C language involves into different processes. Below diagram will help you to understand all the processes that a C program comes across.




```
                VIM - Vi IMproved

                 version 7.4.640
            by Bram Moolenaar et al.
      Modified by <bugzilla@redhat.com>
    Vim is open source and freely distributable

           Sponsor Vim development!
  type  :help sponsor<Enter>    for information

  type  :q<Enter>               to exit
  type  :help<Enter>  or  <F1>  for on-line help
  type  :help version7<Enter>   for version info
```

TEXT EDITOR

VIM

4.1. About Vim

Vim is a text editor that stands for “Vi Improved”. As the name suggests, it is upwards compatible to Vi and it can be used to edit all kinds of plain text documents. It's typically used

to edit program source files and Linux/Unix system configuration files. It also happens to be installed by default on almost every Unix or Unix-clone (i.e. Linux) platform.

Some improvements that Vim has over Vi are multi-level undo, multi windows and buffers, syntax highlighting, command line editing, filename completion, online help, visual selection, etc.. See :help vi_diff.txt for a summary of the differences between Vim and Vi.

While running Vim, help can be obtained from the on-line help system, with the :help command.

Most often Vim is started to edit a single file with the command:

vim filename

More generally, Vim is started with:

vim [options][filelist]

A list of options can be obtained by reading the manual pages on Vim using the command:

man vim

There is an interactive Vim tutorial program that you can work through by using the command:

vimtutor

4.2. Editing Basics:

Invoking Vim

To start Vim on the command line type:

vim <Enter>

You will see the welcome screen in the Vim program appear as shown in the front picture.

Vim Modes: Command and Insert

There are two primary modes in Vim:

Command and Insert. Vim starts in Command mode. You cannot insert text while in Command mode.

To switch from Command mode to Insert mode

type: **i**

The word ‘insert’ should appear at the bottom of the screen indicating you are in Insert mode.

Now you can insert text at the cursor position. Type the following traditional Hello World program into the buffer:

```
#include <iostream>

using namespace std;

int main()

{

cout << "Hello World\n";

return 0;

}
```

After typing the Hello World program we are ready to switch from Insert Mode to Command mode. To enter Command Mode press:

<Esc>

Notice that the word Insert is not at the bottom of the screen. We are now in command mode.

Opening Existing Files and Saving (or Not)

In order to save our work and exit the vim program we type:

:wq hello.cpp

Then press <Enter>

This should return you to your bash shell prompt. From here you can see the file that vi created by using the ls command.

To open hello.cpp in Vim from the bash shell prompt we type:

vim hello.cpp

Now if we want to quit without saving we type:

:q

Then press:

<Enter>

If the file hello.cpp has been edited then Vim will not let you exit without saving.

Open hello.cpp in Vim by typing:

vim hello.cpp

Then type: i

Now type: change

Then press: <Esc> Then type: :q

Then press: <Enter> If you want to exit Vim after editing hello.cpp

then you have to do a forced quit by typing: :q!

To insert another file into the current file at the current cursor position perform the following steps.

Type:

vim hello.cpp

First move the cursor to the bottom of the hello.cpp file and type:

:r .bash_profile

Now, if we edit the file and then want to remove all the edits we have made we type:

:e!

Moving Around

As you may have already found out, when you are in Insert mode you can use the arrow keys to move left, right, up, or down. There are a few tricks for moving through text in Command mode that can be helpful.

Now type:

Do not need!

vim hello.cpp <Enter>

To insert (append) text at the end of a line type A in Command mode. This positions the cursor at the end of the line and switches Vim to Insert mode. Now move your cursor to the

top hello.cpp and type:

A

To insert text at the beginning of a line, type I in Command mode. This positions the cursor at the beginning of the line and switches Vim to Insert mode. Go back to Command mode by pressing:

<Esc>

Then type:

I

Now press:

<Esc>

Now exit your hello.cpp file by typing:

:q <enter>

Now type:

vim /etc/profile <Enter>

To scroll forward through a screen of text type

<Ctrl> f

To scroll backward through a screen of text type

<Ctrl> b

Now close the /etc/profile file by typing:

:q <Enter>

Oops! (Undoing Mistakes)

Like almost every good text editor, there is an undo feature to correct past mistakes in editing your file. Vim keeps a history of your edits and can undo multiple edits.

Now type:

vim hello.cpp <Enter>

Go into Insert mode by typing:

i

Then type: change

Press <Esc>

To undo your last edit type:

u

For every consecutive u typed in command mode anotherof your last edits is undone.

Enter insert mode by typing:

i

Go to your cout statement and change it to:

cout >> "World Hello/n"

Press: <Esc>

To undo all edits on a single line in the file type:

U

This command will only work if your cursor is still on the line you wish to undo. Once your cursor has moved off the line you cannot undo the line with the U command.

Text Searching:

You can search for any string in Vim using a variety of search commands while in Command mode:

Search by Pattern

T o search forward for a pattern o type:

/o

To search for the next instance of o type:

n

To search for the previous instance of o type:

N

T o search backward for a pattern e type:

?e

To search for the next instance of e type:

n

S earch in g by Line Number

Compilers generally display error messages using line numbers. So being able to find specific lines based on the line number is a very

useful feature.

To find the line number of your current cursor position,

press and hold:

Then type:

<Ctrl> g

You should see the file name, line position, document location percent, and column number at the bottom of the screen. To move to a s pecific line in the file, type the line number you want followed b y G in Command mode.

For example:

To move the cursor to line 44 you would type 44G. Move your cursor to the top of hello.cpp and type:

5G

This should have moved your cursor to the cout statement in hello.cpp.

Substitution:

To perform a global substitution that will replace every instance of e to T type:

:%s/e/T/g

T his will replaced all instances of e with T no questions as ked.

If you want to do a global substitution with confirmation on each instance of T with e then type:

:%s/T/e/gc

This will prompt you for every instance of T and ask if you want to substitute it for e.

N ow type:

y

To quit type:

q

Manipulating Text:

Copying

The command to copy a line text is yy in Command mode. To copy multiple lines from the current cursor position enter the number of lines bef ore yy. For example:

To copy 12 lines: 12yy

To copy 12 lines: 12yy

Move your cursor to the main function of the hello.cpp and type:

5yy

The number of lines you are copying should be displayed at the bottom of the screen as yanked lines.

Deleting/Cutting

Cutting lines works exactly like copying lines except replace the use “yy” with “dd”.

Pasting

To paste either copied or cut lines type p in command mode. This will paste the lines under the current position of the cursor. Move your cursor to the bottom of the hello.cpp and type:

p

The number of lines pasted should be displayed at the bottom of the screen. Now we will

revert hello.cpp to it's original content and quit Vim by typing:

:e! <Enter>

Next type:

:q <Enter>

4.3. Recovering from a system crash:

Like most editors, if the system crashes or the secure shell connection gets disconnected there is a way to recover the edit buffer for the file you were last editing. To recover the edit buffer type vim -r hello.cpp at the bash shell prompt.

4.4. Customizing Vim:

You can set Vim options a nd customize Vim's behavior by creating and editing a .vimrc file in your home directory. Here is an example of a nice .vimrc to assist the writing of programs in C++ or C. The nocp option turns off strict vi compatibility. The incsearch option turns on incremental searching. The number option displays the line number to the left of each line and the showmatch option matches brackets {} after closing a block of code. The terse option turns off the message displayed when a search has wrapped around a file. The settings for cinoptions, cinwords, and formatoptions differ from the defaults. The result is to produce a fairly strict “K&R” C formatting style.

COMPILER

GCC

5.1. Programming Languages Supported by GCC

GCC stands for “GNU Compiler Collection”. GCC is an integrated distribution of compilers for several major programming languages. These languages currently include C, C++, Objective-C, Objective-C++, Java, Fortran, and Ada.

The abbreviation GCC has multiple meanings in common use. The current official meaning is “GNU Compiler Collection”, which refers generically to the complete suite of tools. The name historically stood for “GNU C Compiler”, and this usage is still common when the emphasis is on compiling C programs. Finally, the name is also used when speaking of the language-independent component of GCC: code shared among the compilers for all supported languages.

The language-independent component of GCC includes the majority of the optimizers, as well as the “back ends” that generate machine code for various processors. The part of a compiler that is specific to a particular language is called the “front end”. In addition to the front ends that are integrated components of GCC, there are several other front ends that are maintained separately. These support languages such as Pascal, Mercury, and COBOL. To use these, they must be built together with GCC proper. Most of the compilers for languages other than C have their own names. The C++ compiler is G++, the Ada compiler is GNAT, and so on. When we talk about compiling one of those languages, we might refer to that compiler by its own name, or as GCC. Either is correct. Historically, compilers for many languages, including C++

and Fortran, have been implemented as “preprocessors” which emit another high level language such as C. None of the compilers included in GCC are implemented this way; they all generate machine code directly. This sort of preprocessor should not be confused with the C preprocessor, which is an integral feature of the C, C++, Objective-C and Objective-C++ languages.

5.2. C language

GCC supports three versions of the C standard, although support for the most recent version is not yet complete. The original ANSI C standard (X3.159-1989) was ratified in 1989 and published in 1990. This standard was ratified as an ISO standard (ISO/IEC 9899:1990) later in 1990. There were no technical differences between these publications, although the sections of the ANSI standard were renumbered and became clauses in the ISO standard. This standard, in both its forms, is commonly known as C89, or occasionally as C90, from the dates of ratification. The ANSI standard, but not the ISO standard, also came with a Rationale document. To select this standard in GCC, use one of the options ‘-ansi’, ‘-std=c90’ or ‘-std=iso9899:1990’; to obtain all the diagnostics required by the standard, you should also specify ‘-pedantic’ (or ‘-pedantic-errors’ if you want them to be errors rather than warnings). See Section 3.4 [Options Controlling C Dialect], page 28. Errors in the 1990 ISO C standard were corrected in two Technical Corrigenda published in 1994 and 1996. GCC does not support the uncorrected version. An amendment to the 1990 standard was published in 1995. This amendment added digraphs and __STDC_VERSION__ to the language, but otherwise concerned the library. This amendment is commonly known as AMD1; the amended standard is sometimes known as C94 or C95. To select this standard in GCC, use the option ‘-std=iso9899:199409’ (with, as for other standard versions, ‘-pedantic’ to receive all required diagnostics). A new edition of the ISO C standard was published in 1999 as ISO/IEC 9899:1999, and is commonly known as C99. GCC has incomplete support for this standard version; see http://gcc.gnu.org/gcc-4.5/c99status.html for details. To select this standard, use ‘-std=c99’ or ‘-std=iso9899:1999’. (While in development, drafts of this standard version were referred to as C9X.)

Errors in the 1999 ISO C standard were corrected in three Technical Corrigenda published in 2001, 2004 and 2007. GCC does not support the uncorrected version. By default, GCC provides some extensions to the C language that on rare occasions conflict with the C standard. See Chapter 6 [Extensions to the C Language Family], page 277. Use of the ‘-std’

options listed above will disable these extensions where they conflict with the C standard version selected. You may also select an extended version of the C language explicitly with '-std=gnu90' (for C90 with GNU extensions) or '-std=gnu99' (for C99 with GNU extensions). The default, if no C language dialect options are given, is ' std=gnu90'; this will change to '-std=gnu99' in some future release when the C99 support is complete.

Some features that are part of the C99 standard are accepted as extensions in C90 mode. The ISO C standard defines (in clause 4) two classes of conforming implementation. A conforming hosted implementation supports the whole standard including all the library facilities; a conforming freestanding implementation is only required to provide certain library facilities: those in <float.h>, <limits.h>, <stdarg.h>, and <std also those in <iso646.h>; and in C99, also those in <stdbool.h> and <stdint.h>. In addition, complex types, added in C99, are not required for freestanding implementations. The standard also defines two environments for programs, a freestanding environment, required of all implementations and which may not have library facilities beyond those required of freestanding implementations, where the handling of program startup and termination are implementation-defined, and a hosted environment, which is not required, in which all the library facilities are provided and startup is through a function int main (void) or int main (int, char *[]). An OS kernel would be a freestanding environment; a program using the facilities of an operating system would normally be in a hosted implementation. GCC aims towards being usable as a conforming freestanding implementation, or as the compiler for a conforming hosted implementation. By default, it will act as the compiler for a hosted implementation, defining __STDC_HOSTED__ as 1 and presuming that when the names of ISO C functions are used, they have the semantics defined in the standard. To make it act as a conforming freestanding implementation for a freestanding environment, use the option '-ffreestanding'; it will then define __STDC_HOSTED__ to 0 and not make assumptions about the meanings of function names from the standard library, with exceptions noted below. To build an OS kernel, you may well still need to make your own arrangements for linking and startup. See Section 3.4 [Options Controlling C Dialect], page 28. GCC does not provide the library facilities required only of hosted implementations, nor yet all the facilities required by C99 of freestanding implementations; to use the facilities of a hosted environment, you will need to find them elsewhere (for example, in the GNU C library).

Most of the compiler support routines used by GCC are present in 'libgcc', but there are a few exceptions. GCC requires the freestanding environment provide memcpy, memmove, memset

and memcmp. Finally, if __builtin_trap is used, and the target does not implement the trap pattern, then GCC will emit a call to abort.

5.3. SYNOPSIS

gcc [-c|-S|-E] [-std=standard]

[-g] [-pg] [-Olevel]

[-Wwarn...] [-Wpedantic]

[-Idir...] [-Ldir...]

[-Dmacro[=defn]...] [-Umacro]

[-foption...] [-mmachine-option...]

[-o outfile] [@file] infile...

Only the most useful options are listed here; see below for the remainder. g++ accepts mostly the same options as gcc.

5.4. DESCRIPTION

When you invoke GCC, it normally does preprocessing, compilation, assembly and linking. The "overall options" allow you to stop this process at an intermediate stage. For example, the -c option says not to run the linker. Then the output consists of object files output by the assembler.

Other options are passed on to one stage of processing. Some options control the preprocessor and others the compiler itself. Yet other options control the assembler and linker; most of these are not documented here, since you rarely need to use any of them.

Most of the command-line options that you can use with GCC are useful for C programs; when an option is only useful with another language usually C++), the explanation says so explicitly. If the description for a particular option does not mention a source language, you can use that option with all supported languages.

The gcc program accepts options and file names as operands. Many options have multi-

letter names; therefore multiple single-letter options may not be grouped: -dv is very different from -d -v.

You can mix options and other arguments. For the most part, the order you use doesn't matter. Order does matter when you use several options of the same kind; for example, if you specify -L more than once, the directories are searched in the order specified. Also, the placement of the -l option is significant.

Many options have long names starting with -f or with -W---for example, -fmove-loop-invariants, -Wformat and so on. Most of these have both positive and negative forms; the negative form of -ffoo is -fno-foo. This manual documents only one of these two forms, whichever one is not the default.

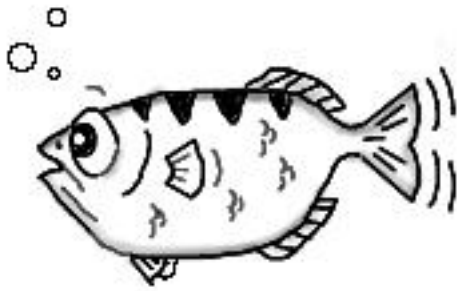
OVERALL OPTIONS

-c -S -E -o file -no-canonical-prefixes -pipe -pass-exit-codes -x language -v -### --help[=class[,...]] --target-help

--version -wrapper @file -fplugin=file -fplugin-arg-name=arg -fdump-ada-spec[-slim] -fada-spec-parent=unit -fdump-go-spec=file

DEBUGGER

GDB: The GNU Project Debugger



6.1. gdb - The GNU Debugger

GDB, the GNU Project debugger, allows you to see what is going on `inside' another program while it executes -- or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.

6.2. SYNOPSIS

gdb [-help] [-nh] [-nx] [-q] [-batch] [-cd=dir] [-f] [-b bps]

[-tty=dev] [-s symfile] [-e prog] [-se prog] [-c core] [-p procID]

[-x cmds] [-d dir] [prog|prog procID|prog core]

6.3. DESCRIPTION

The purpose of a debugger such as GDB is to allow you to see what is going on "inside" another program while it executes -- or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

· Start your program, specifying anything that might affect its behavior.

· Make your program stop on specified conditions.

· Examine what has happened, when your program has stopped.

· Change things in your program, so you can experiment with correcting the effects of one

bug and go on to learn about another.

You can use GDB to debug programs written in C, C@t{++}, Fortran and Modula-2. GDB is invoked with the shell command "gdb". Once started, it reads commands from the terminal until you tell it to exit with the GDB command "quit". You can get online help from GDB itself by using the command "help". You can run "gdb" with no arguments or options; but the most usual way to start GDB is with one argument or two, specifying an executable program as the argument:

gdb program

You can also start with both an executable program and a core file specified:

gdb program core

You can, instead, specify a process ID as a second argument, if you want to debug a running process:

gdb program 1234

gdb -p 1234

would attach GDB to process 1234 (unless you also have a file named 1234; GDB does check for a core file first). With option -p you can omit the program filename. Here are some of the most frequently needed GDB commands:

break [file:]functiop

Set a breakpoint at function (in file).

run [arglist]

Start your program (with arglist, if specified).

bt Backtrace: display the program stack.

print expr

Display the value of an expression.

c Continue running your program (after stopping, e.g. at a breakpoint).

next

Execute next program line (after stopping); step over any function calls in the line.

edit [file:]function

look at the program line where it is presently stopped.

list [file:]function

type the text of the program in the vicinity of where it is presently stopped.

step

Execute next program line (after stopping); step into any function calls in the line.

help [name]

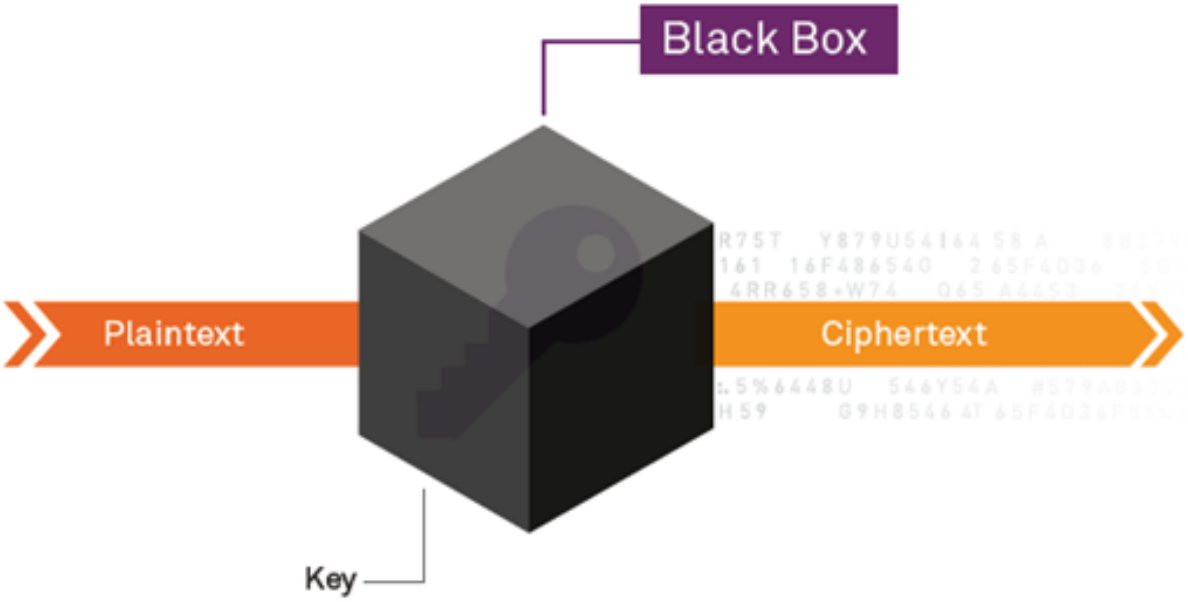
Show information about GDB command name, or general information about using GDB.

quit

ENDECS

CRYPTOGRAPHY TOOL

…….encryption decryption system



1.1. INTRODUCTION

This application has been designed on the theory of multiple data compression and encryption. A breief introduction of this technique is presented below : -

1.2. Multiple Data Compression and Encryption using Iterative Technique

Abstract:

Encoding data to take up less storage space and less bandwidth for transmission. Digital data are compressed by finding repeatable patterns of binary 0s and 1s. The more patterns can be found, the more the data can be compressed. Text can typically be compressed to approximately 40% of its original size, and graphics files from 20% to 90%. Some files compress very little. It depends entirely on the type of file and compression algorithm used.

Data compression is particularly useful in communications because it enables devices to transmit or store the same amount of data in fewer bits. There are a variety of data compression techniques, but only a few have been standardized.

Encryption is the conversion of data into a form, called a ciphertext, that cannot be easily understood by unauthorized people. Decryption is the process of converting encrypted data back into its original form, so it can be understood.

Objective:

Compress the textual data / text files and reduce the size of the files, Also Encrypt the compressed file so that the unintended user can not understand it.

Synopsis:

Data compression, **source coding**,or **bit-rate reduction** involves encoding information using fewer bits than the original representation. The Compression Technique used is Lossless compression that reduces bits by identifying and eliminating. No information is lost in lossless compression. The process of reducing the size of a data file is popularly referred to as data compression, although its formal name is source coding i.e. coding done at the source of the data, before it is stored or transmitted.

We identify the number of unique characters in the source file. Then depending on the number of unique characters we assign the code of certain length (less then the Byte size) to each character.

These smaller codes are saved in the compressed file.

This process could be repeated a to achieve greater depth of compression.

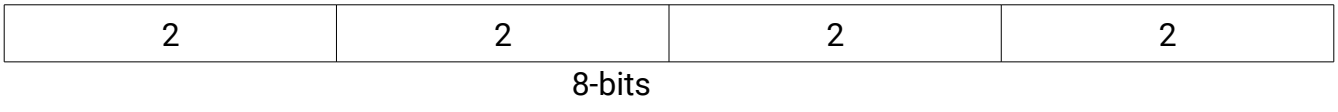
languages and Tools used for this project would be: Linux Operation System, C programming, gcc compiler, gdb debugger, make, makefile utilities for files management, RCS source code management tool, CVS Server.

In general, a keyboard works on the basis of ASCII codes. These codes of of 8-bits, it means we can use 2^8 distinct characters simultaneously. If a file have only 4, 8, 16, 32, 64, 128, then in those cases, we will need only 2, 3, 4, 5, 6 ,7 respectively code-lengths to represent every character of the file. So, we can reduce the file up to 80%. Also there is another advantages, the compressed file will also be encrypted so data will be protected from any unfair means.

1.3. SOURCE CODE STRUCTURE OF THE APPLICATION :

There is a main file which will transfer the program control to the desired task i.e., encryption or decryption. This main program does not terminate automatically. It run and wait for the entry of the user. When encryption process is selected then the program control goes to the encryption source code. Then a file name of the file containing the plain text will be asked. Application will open the file and copy the data into a temporary buffer. Now, another file will be created containing distinct characters of the input file. This file is the encryption key. Now, the length of every character is depends upon the number of distinct characters in the encryption key. The code-length for every character of the file will be 2, 3, 4, 5, 6, or 7 if there are less than 4, 8, 16, 32, 64 or 127 distinct characters in the encryption key.

If code-length of the character is 2-bits then in a 8-bits architecture of the buffer (used for ASCII codes) we can adjust 4 characters in only one ASCII character space. As following :



8-bits

In this way, we can compress the original file up to 75%.

if the code-length of the character is 3-bit then we can adjust the characters as following(8 characters are adjusted in 3 characters space) : -

3	3	2

8-bits

1	3	3	1

8-bits

2	3	3

8-bits

If the code-length of the character is 4-bits then two characters can be adjusted into only one character space.

4	4

8-bits

if the code-length of the character if 5-bits then the adjustment of the character will be as following : -

5	3

8-bits

2	5	1

8-bits

4	4

8-bits

1	5	2

8-bits

3	5

8-bits

Here, 8 character are adjusted in only 5 character space.

If the length of the character is 6-bit then the arrangement of the characters will be as following :-

6	2

8-bits

4	4

8-bits

2	6

8-bits

For the file for length of every character 7-bits, arrangement will be as following :-

7	1

8-bits

6	2

8-bits

5	3

8-bits

4	4

8-bits

3	5

8-bits

2	6

8-bits

1	7

8-bits

There is no need to compress the file having the characters of the length 8-bits. Next interesting thing is that we have only 127 distinct characters used in the computer technology. So, we can definitely encrypt any type of file.

Let's move towards the decryption.

In the decryption, the opposite operation will be performed with the encrypted data. Like in case of 3-bits decryption, every 3-bits from the encrypted will be fetched and converted into 8-bit character and the ASCII value will be stored in the file for the plain text.

1.4. User's Guide : -

This application ENDECS can be executed only on the Linux operating system. Now, copy this application to any directory of the file system ext4 or ext2 of the system file. Give it the permission for execution by the following command :

```
chmod +x ENDECS
```

now run the application as following :

```
./ENDECS
```

you will get the following window :

```
ENDEC
-----------------------------------------------------------------------------------------------------------
::TERMS & CONDITIONS::
-----------------------------------------------------------------------------------------------------------
1. This application is protected under the copyright law.
2. Any type of copy of the source code or reverse engieening is not permitted.
3. This application is half open source you can use this application free but cannot view the source code.
4. The  success rate of this application is 75 percent. In case of failure the original file will be protected.
-----------------------------------------------------------------------------------------------------------
Do you want to countinue (y/n)? y
```

now if you are agree with the terms and conditions give the input :'y' <enter>

Now the screen will be like following :

```
-------------------------------------------------
-------------------------------------------------
::OPERATIONS::
1. Compression & Encryption
2. Decompression & Decryption
0. Exit the application
-------------------------------------------------
Enter the task ... !!
```

Enter you choice and press <enter> to be continue.

Let us encrypt a file first.

Enter option 1 and the screen will be like following :

```
Enter the filename you want to compress or encrypt :
```

Enter the file name. In our case the name is ffc (file for compression)

After giving the file name you will have the following situation :

```
'ffc' is opened successfully on the file descripter : 3
File ffc
Data Stucture & Algorithm Designing
If a person is differentiating between hardware and software,
definitely, he is not a computer engineer.
!@#$%^&*()_+~`1234567890-=][\|;:'",><./?
abcdefghijklmnopqrstwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ



Number of charaters : -20

Cryptographic Key has been successfully written in the file CKEY.
Cryptographic key
Dat Sucre&Algoihmsn
Ifpdbw,y.!@#$%^*()_+~`1234567890-=][\|;:'"></?jkqxzBCEFGHJKLMNOPQRTUVWXYZ
Number of characters : 93
The length of the code of every charater is 7 bits.
Compressed file 'CFILE' has been successfully created.

------------------------------------------------------------------------
::OPERATIONS::
1. Compression & Encryption
2. Decompression & Decryption
0. Exit the application
------------------------------------------------------------------------
Enter the task ... !!
```

The file has been successfully encrypted and stored in the file 'CFILE' and length of the character is 7-bits. The encryption key you can see in the screenshot, has been written in a file CKEY.

Use VIM editor to check the cipher text : vim CFILE <enter>

```
^@^D^P^Pa^A^E^L^H(q^@Å<83>^T,`Ðã<81>^O ^L^@<82>#<86>^R^\Ha2<85>A<81>^FX@r#I^C^\D^YqÅJ<88>^N <90>!ÁA^N$0^Y<81>^@<8c><88>^PH^Xð!É<99>^B^\@0$<8b><83>"4¨
# C<88>4L¸<82>£<89>^N^D Y³@Ç<88>^F8<88>2CA^C^B^L0Ò^E<82><82>^P^\^X<82>C^G^R^P 9Åg0^_@<85>^P<94>i^R¦N¡J¥k^V®^Á<8b>&m^Z¶náË§o^^<9a>|üä^H$Å<98>^L\AQ<83>
ÇB<86>,<81>!¥¢^G"^HÌSq<89><8a><8f> ^D<99>R¤^T<9b>:}
4©Ó³^Rµ{^Vm[¸L<99>?à
~
```

You can see the encrypted data. “Are you able to understand ?” Perhaps No. So am I.. Let us go to decry-pt the cipher text.

As we know, for decryption two things are necessary :

1. Cipher text

2. Encryption Key

We have CFILE (cipher text file) and CKEY (encryption key) are two file which can be used to get the plain text. But how ??

Back to the application, run it again and choose the option for decryption and decompression i.e., '2'.

Now, you will be asked for two files cipher text and encryption key.

Give the name of these of these files. After giving the both file name, you will get the following screen :

```
Enter the task ... !!   2
------------------------------------------------------------
Enter the filename which you want to decrypt:   CFILE
Enter the filename of cryptographic key:        CKEY

'CFILE' is opened successfully on the file descripter : 3

'CKEY' is opened successfully on the file descripter : 3
Number of characters in the file is 94.
Code-length of every character of the file is 7-bits
The plain text is successfully written in the file 'DFILE'.

------------------------------------------------------------
```

The plain text has been successfully written in the file 'DFILE'. Now check out the DFILE file. vim DFILE <enter>

```
Data Stucture & Algorithm Designing
If a person is differentiating between hardware and software,
definitely, he is not a computer engineer.
!@#$%^&*()_+~`1234567890-=][\|;:'",><./?
abcdefghijklmnopqrstwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

we got the plain text.

Now exit from the application by giving the input 0 :

```
------------------------------------------------
Enter the task ... !!   0
------------------------------------------------
Application is going to be terminated
Thank you for using ENDEC cryptography tool  !!
------------------------------------------------
```