India's Most Popular
Online IT Championship

28 States, Hundreds of School,
Thousands of Students, One Platform

Preparation & Reference Book

Category - Senior

Class : X

www.itolympiad.in

CITIS Infotech : Knowledge Partner

MESSAGE

Dear Learner

Greetings.

I.T. Olympiad has gone from pillar to post in last few years. Thousands of students of all state boards, CBSE, CISCE and International Boards are appearing in this online championship ever year.

The objective of this competition is to encourage the students towards computing at an early age. Fortunately Indian schools are giving good stress on computer education in their curriculums and the results of such motivation are also visible; however, the awareness of Free and Open source computing philosophy is not much discussed in India at the school level.

In It's National Policy on ICT in School education 2012, Govt. of India clearly mention the policy goal is to promote universal, equitable, open and free access to a state of the art ICT and ICT-enabled tools and resource to all the students and teachers. Under section 6.3 Software paragraphs 6.3.1 it clearly mentions Free and Open Source Software- Operating System and software application will be preferred in order to expand the range of learning, creation, and sharing.

Keeping the above objective in mind this new curriculum of IT Olympiad Cubs(Junior Category) has lessons on FOSS based Operating System Ubuntu, GIMP, Email Introduction, Blogging, accessing various E-Services and some of the most popular open source applications which students can use freely. We hope that students will be full of new ideas and skills after mastering the topics presented in this book.

Please feel free to share your feedback with us.

With best wishes

Prof. Dr. R. V. Aacharya & Team of IT Olympiad

TABLE OF CONTENTS

1.Indian IT Industry

One of the faster growing sector of today's India is IT. According to wikipedia on IT Industry in India- Information Technology in India is an industry comprising of two noteworthy segments, IT administrations(services) and business process outsourcing (BPO). The area has expanded its commitment to India's GDP from 1.2% in 1998 to 7.5% in 2012. According to NASSCOM, the sector aggregated revenues of US$147 billion in 2015, where export revenue stood at US$99 billion and domestic at US$48 billion, growing by over 13%.

India's present leader Narendra Modi has begun 'The Digital India' venture with a vision to transform India into a digitally empowered society and knowledge economy.

How it started:

Shri V. Rajaraman IIT Bangalore mentions in his paper 'HISTORY OF COMPUTING IN INDIA' that The computer age in India started in 1955 with the establishment of HEC-2M (a computer designed by A.D.Booth in England) at the Indian Statistical Institute (ISI) at Calcutta (now Kolkata). That year a group headed by R.Narasimhan began planning and creating a computer at the Tata Institute of Fundamental Research (TIFR) at Bombay (now Mumbai) 2000 km away on the west bank of India. In 1955 just a couple of dozen researchers and specialists in India thought about computers.

It was nothing more than a number crunching machine and was huge in size. The dimensions of this machine were 10 ft in length, 7 ft in breadth and 6 ft in height. It played a critical role in formulating annual and five-year plans by the planning commission, and in top-secret projects of India's nuclear program. Moreover, it went on to turn out India's first generation of computer professionals. It was at least ten thousand times slower in solving even simple problems than today's machines. But it set the stage for the development of computers in India.

The HEC-2M also played a pivotal role in the statistical data processing that formed the bedrock of the five-year plans. India's weather forecasting model, too, based on statistical analysis of meteorological data, was developed on it. Most importantly, the same machine was used to design the next generation of computers, including India's first indigenous computer, the 'TIFRAC' (or Tata Institute of Fundamental Research Automatic Computer), in 1962.

From that point in time, today India has come a long way. Today almost every office desk in India has a PC and government's IT policies shows its sincere efforts to reach out to every village in the country. For a nation like India which is geographically big and culturally and linguistically so varied, computer technology has proved to be a great tool of overall development. Successful efforts are made at government as well as non-government level to use this technology for the benefit of the Indian society.

* * *

Exercise : A) Prepare a note on the history of I.T. Industry world wide.

B) Collect the latest figure and data showing the progress of India IT Industry in past to years.

2. History of Software

An interesting story of software development:

Software Programming (development) can be defined as programmed instructions stored in the memory of stored-program digital computers for execution by the processor. The design for what would have been the first piece of software was written by Ada Lovelace in the 19th century.

Birth of "Software" and the Interactive Minicomputer

According to Jeffery R. Yost, the term "software" was created in the late 1950s and was soon adopted throughout the industry (2005). Coined by statistician John Tukey, the term became a catchall, user-friendly term for the work of computer programmers who were using terminology ranging from "computer program" to "code."

The America Heritage New Dictionary of Cultural Literacy describes software as "the programs and instructions that run a computer, as opposed to the actual physical machinery and devices that compose the hardware." Meanwhile, The Free On-Line Dictionary of Computing adds that software is divided into two primary types: system software and program applications.

System software includes general program execution processes such as compilers and, most recognizably, the disk operating system (DOS), which has evolved in form in IBM PC-style computers within the last two decades from the ubiquitous Microsoft DOS prompt (MS-DOS) to stylish Windows-based platforms from Microsoft 2000 to Windows Vista.

Similarly, Apple has seen countless new releases from the Apple DOS 3.1 of 1977 to the OS X series of recent years. Program applications include everything else, from gaming to multimedia to scientific applications. Finally, software combines lines of source code written by humans with the work of compilers and assemblers in executing machine code (Info Source : Dictionary.com).

As a profession, software development has its roots in the 1960s. Today software development business run on the quality of its development which mainly consist various components such as its maintenance, stability, speed, testing, readability, size, cost, security and number of issues or 'bugs' etc.

* * *

3. The Copyright and Proprietary Software Business

The world's first copyright law was the Statute of Anne, ordered in England in 1710. This Act presented surprisingly the idea of the creator of a work being the proprietor of its copyright, and laid out settled terms of assurance. Taking after this Act, copyrighted works were required to be kept at particular copyright libraries, and enlisted at Stationers' Hall. There was no automatic copyright security for unpublished works.

The software Copyright

Computer software was one of the first electronic information technology products that penetrated the domain of intellectual property.

There was little requirement for copyright (or patent) security for early computer programs. There were couple of computers, and most programming was exceptionally produced for in-house applications. It wasn't until the early 1960s that computer programs were being actively marketed by a software industry besides the computer manufacturers. Before broadly promoted programming, it was anything but difficult to ensure by an agreement or permit understanding any computer program that was being showcased.

United States practice

According to Wikipedia, Copyright protection attaches to "original works of authorship fixed in any tangible medium of expression, now known or later developed, from which they can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device."17 U.S.C.A. § 102. Copyright functions by granting the author the right to exclude others. Copyright protects:

- literary works
- musical works (& accompanying words)
- dramatic works (& accompanying music)
- pantomimes and choreographed works
- pictorial, graphic, & sculptural works
- motion pictures & other audiovisual works
- sound recordings
- architectural works

+ compilations and derivative works – 17 USC § 103(a).
In the United States, computer programs are literary works, under the definition in the Copyright Act, 17 U.S.C. § 101.

Indian Copyright Policies: According to Govt. of India's copyright definiation mentioned on its official website http://copyright.gov.in - "Copyright is a right given by the law to creators of literary, dramatic, musical and artistic works and producers of cinematograph films and sound recordings. In fact, it is a bundle of rights including, inter alia, rights of reproduction, communication to the public, adaptation and translation of the work. There could be slight variations in the composition of the rights depending on the work.

Copyright ensures certain minimum safeguards of the rights of authors over their creations, thereby protecting and rewarding creativity. Creativity being the keystone of progress, no civilized society can afford to ignore the basic requirement of encouraging the same. Economic and social development of a society is dependent on creativity. The protection provided by copyright to the efforts of writers, artists, designers, dramatists, musicians, architects and producers of sound recordings, cinematograph films and computer software, creates an atmosphere conducive to creativity, which induces them to create more and motivates others to create."

The copyright in India has travelled a long way since it was introduced during the British rule. The first law on copyright was enacted in the year 1847 by the then Governor General of India. When Copyright Act 1911 came into existence in England, it became automatically applicable to India, being India an

integral part of British Raj. This act was in force in the country until after independence when a new copyright act (the Act of 1957) came into effect in 1958. Thereafter the Act has undergone many amendments. The latest in the series is the 1994 Amendment, which came into force in May 1995.

Proprietary software

Proprietary software as described by Wikipedia is computer software for which the software's publisher or another person retains intellectual property rights—usually copyright of the source code, but sometimes patent rights.

Examples of proprietary software include Microsoft Windows, Adobe Flash Player, PS3 OS, iTunes, Adobe Photoshop, Google Earth, Mac OS X, Skype, WinRAR, Oracle's version of Java and some versions of Unix.

Software distributions considered as proprietary may in fact incorporate a "mixed source" model including both free and non-free software in the same distribution.] Most if not all so-called proprietary UNIX distributions are mixed source software, bundling open-source components like BIND, Sendmail, X Window System, DHCP, and others along with a purely proprietary kernel and system utilities

* * *

4. End-user license agreement (EULA)

As mentioned in Wikipedia, In proprietary software, an end-user license agreement (EULA) or software license agreement is the contract between the licensor and purchaser, establishing the purchaser's right to use the software. The license may define ways under which the copy can be used, in addition to the automatic rights of the buyer including the first sale doctrine and 17 U.S.C. § 117 (freedom to use, archive, re-sale, and backup).

Many form contracts are only contained in digital form, and only presented to a user as a click-through where the user must "accept". As the user may not see the agreement until after he or she has already purchased the software, these documents may be contracts of adhesion.

Software companies often make special agreements with large businesses and government entities that include support contracts and specially drafted warranties. Some end-user license agreements accompany shrink-wrapped software that is presented to a user sometimes on paper or more usually electronically, during the installation procedure. The user has the choice of accepting or rejecting the agreement. The installation of the software is conditional to the user clicking a button labelled "accept".

Many EULAs assert extensive liability limitations. Most commonly, an EULA will attempt to hold harmless the software licensor in the event that the software causes damage to the user's computer or data, but some software also proposes limitations on whether the licensor can be held liable for damage that arises through improper use of the software (for example, incorrectly using tax preparation software and incurring penalties as a result).

One case upholding such limitations on consequential damages is *M.A. Mortenson Co. v. Timberline Software Corp., et al.* Some EULAs also claim restrictions on venue and applicable law in the event that a legal dispute arises.

Some copyright owners use EULAs in an effort to circumvent limitations the applicable copyright law places on their copyrights (such as the limitations in sections 107–122 of the United States Copyright Act), or to expand the scope of control over the work into areas for which copyright protection is denied by law (such as attempting to charge for, regulate or prevent private performances of a work beyond a certain number of performances or beyond a certain period of time).

Such EULAs are, in essence, efforts to gain control, by contract, over matters upon which copyright law precludes control. This kind of EULAs concurs in aim with DRM and both may be used as alternate methods for widening control over software. In disputes of this nature in the United States, cases are often appealed and different circuit courts of appeal sometimes disagree about these clauses. This provides an opportunity for the U.S. Supreme Court to intervene, which it has usually done in a scope-limited and cautious manner, providing little in the way of precedent or settled law.

Criticism

One common criticism of end-user license agreements is that they are often far too lengthy for users to devote the time to thoroughly read them. In March 2012, the PayPal end-user license agreement was 36,275 words long and in May 2011 the iTunes agreement was 56 pages long. News sources reporting these findings asserted that the vast majority of users do not read the documents because of their length.

Several companies have parodied this belief that users do not read the end-user-license agreements by adding unusual clauses, knowing that few users will ever read them. As an April Fool's Day joke, Gamestation added a clause stating that users who placed an order on April 1, 2010 agreed to irrevocably give their soul to the company, which 7,500 users agreed to.

Although there was a checkbox to exempt out of the "immortal soul" clause, few users checked it and thus Gamestation concluded that 88% of their users did not read the agreement.The program PC Pitstop included a clause in their end-user license agreement stating that anybody who read the clause and contacted the company would receive a monetary reward, but it took four months and over 3,000 software downloads before anybody collected it. During the installation of version 4 of the Advanced Query Tool the installer measured the elapsed time between the appearance and the acceptance of the end-user license agreements to calculate the average reading speed. If the agreements were accepted fast enough a dialog window “congratulated” the users to their absurdly high reading speed of several hundred words per second. *South Park* parodied this in the episode "HumancentiPad", where Kyle had neglected to read the terms of service for his last iTunes update and therefore inadvertently agreed to have Apple employees experiment upon him.

End-user license agreements have also been criticized for containing terms that impose onerous obligations on consumers. For example, Clickwrapped, a service that rates consumer companies according to how well they respect the rights of users, reports that they increasingly include a term that prevents a user from suing the company in court.

* * *

Exercise : A) Try to go through the end user license Agreement of Windows X Par8 and prepare a summary of the agreement

B) Compare EULA with GNU-GPL license.

5. Introduction of Free Software

The Free Software Definition

As mentioned on Free Software Foundation, Inc. page www.gnu.org
The free software definition presents the criteria for whether a particular software program qualifies as free software. From time to time we revise this definition, to clarify it or to resolve questions about subtle issues.

“Free software” means software that respects users' freedom and community. Roughly, it means that **the users have the freedom to run, copy, distribute, study, change and improve the software**. Thus, “free software” is a matter of liberty, not price. To understand the concept, you should think of “free” as in “free speech,” not as in “free milk”. We sometimes call it “libre software,” borrowing the French or Spanish word for “free” as in freedom, to show we do not mean the software is gratis.

We campaign for these freedoms because everyone deserves them.

With these freedoms, the users (both individually and collectively) control the program and what it does for them. When users don't control the program, we call it a “nonfree” or “proprietary” program. The nonfree program controls the users, and the developer controls the program; this makes the program an instrument of unjust power.

The four essential freedoms

A program is free software if the program's users have the four essential freedoms:

- The freedom to run the program as you wish, for any purpose **(freedom 0).**
- The freedom to study how the program works, and change it so it does your computing as you wish **(freedom 1).** Access to the source code is a precondition for this.
- The freedom to redistribute copies so you can help your neighbor **(freedom 2).**
- The freedom to distribute copies of your modified versions to others **(freedom 3).** By doing this you can give the whole community a chance to benefit from your changes. Access to the source code is a precondition for this.

A program is free software if it gives users adequately all of these freedoms. Otherwise, it is nonfree. While we can distinguish various nonfree distribution schemes in terms of how far they fall short of being free, we consider them all equally unethical. In any given scenario, these freedoms must apply to whatever code we plan to make use of, or lead others to make use of.

For instance, consider a program A which automatically launches a program B to handle some cases. If we plan to distribute A as it stands, that implies users will need B, so we need to judge whether both A and B are free. However, if we plan to modify A so that it doesn't use B, only A needs to be free; B is not pertinent to that plan.

“Free software” does not mean “noncommercial”. A free program must be available for commercial use, commercial development, and commercial distribution. Commercial development of free software is no longer unusual; such free commercial software is very important. You may have paid money to get copies of free software, or you may have obtained copies at no charge. But regardless of how you got your copies, you always have the freedom to copy and change the software, even to sell copies.

The rest of this page clarifies certain points about what makes specific freedoms adequate or not.

The freedom to run the program as you wish

The freedom to run the program means the freedom for any kind of person or organization to use it on any kind of computer system, for any kind of overall job and purpose, without being required to communicate about it with the developer or any other specific entity.

In this freedom, it is the *user's* purpose that matters, not the *developer's* purpose; you as a user are free to run the program for your purposes, and if you distribute it to someone else, she is then free to run it for her purposes, but you are not entitled to impose your purposes on her.

The freedom to run the program as you wish means that you are not forbidden or stopped from doing so. It has nothing to do with what functionality the program has, or whether it is useful for what you want to do.

The freedom to study the source code and make changes

In order for freedoms 1 and 3 (the freedom to make changes and the freedom to publish the changed versions) to be meaningful, you must have access to the source code of the program. Therefore, accessibility of source code is a necessary condition for free software.

Obfuscated “source code” is not real source code and does not count as source code.

Freedom 1 includes the freedom to use your changed version in place of the original. If the program is delivered in a product designed to run someone else's modified versions but refuse to run yours — a practice known as “tivoization” or “lockdown”, or (in its practitioners' perverse terminology) as “secure boot” — freedom 1 becomes an empty pretense rather than a practical reality. These binaries are not free software even if the source code they are compiled from is free.

One important way to modify a program is by merging in available free subroutines and modules. If the program's license says that you cannot merge in a suitably licensed existing module — for instance, if it requires you to be the copyright holder of any code you add — then the license is too restrictive to qualify as free.

Whether a change constitutes an improvement is a subjective matter. If your right to modify a program is limited, in substance, to changes that someone else considers an improvement, that program is not free.

The freedom to redistribute if you wish: basic requirements

Freedom to distribute (freedoms 2 and 3) means you are free to redistribute copies, either with or without modifications, either gratis or charging a fee for distribution, to anyone anywhere. Being free to do these things means (among other things) that you do not have to ask or pay for permission to do so.

You should also have the freedom to make modifications and use them privately in your own work or play, without even mentioning that they exist. If you do publish your changes, you should not be required to notify anyone in particular, or in any particular way.

Freedom 3 includes the freedom to release your modified versions as free software.

A free license may also permit other ways of releasing them; in other words, it does not have to be a copyleft license. However, a license that requires modified versions to be nonfree does not qualify as a free license.

The freedom to redistribute copies must include binary or executable forms of the program, as well as source code, for both modified and unmodified versions. (Distributing programs in runnable form is necessary for conveniently installable free operating systems.) It is OK if there is no way to produce a binary or executable form for a certain program (since some languages don't support that feature), but you must have the freedom to redistribute such forms should you find or develop a way to make them.

Contract-based licenses

Most free software licenses are based on copyright, and there are limits on what kinds of requirements can be imposed through copyright. If a copyright-based license respects freedom in the ways described above, it is unlikely to have some other sort of problem that we never anticipated (though this does happen occasionally). However, some free software licenses are based on contracts, and contracts can impose a much larger range of possible restrictions. That means there are many possible ways such a license could be unacceptably restrictive and nonfree.

We can't possibly list all the ways that might happen. If a contract-based license restricts the user in an unusual way that copyright-based licenses cannot, and which isn't mentioned here as legitimate, we will have to think about it, and we will probably conclude it is nonfree.

Why Software Should Be Free : An essay by Richard Stallman
Introduction

The existence of software inevitably raises the question of how decisions about its use should be made. For example, suppose one individual who has a copy of a program meets another who would like a copy. It is possible for them to copy the program; who should decide whether this is done? The individuals involved? Or another party, called the "owner"?

Software developers typically consider these questions on the assumption that the criterion for the answer is to maximize developers' profits. The political power of business has led to the government adoption of both this criterion and the answer proposed by the developers: that the program has an owner, typically a corporation associated with its development.

I would like to consider the same question using a different criterion: the prosperity and freedom of the public in general.

This answer cannot be decided by current law—the law should conform to ethics, not the other way around. Nor does current practice decide this question, although it may suggest possible answers. The only way to judge is to see who is helped and who is hurt by recognizing owners of software, why, and how much. In other words, we should perform a cost-benefit analysis on behalf of society as a whole, taking account of individual freedom as well as production of material goods.

In this essay, I will describe the effects of having owners, and show that the results are detrimental. My conclusion is that programmers have the duty to encourage others to share, redistribute, study, and improve the software we write: in other words, to write “free” software.(1)

How Owners Justify Their Power

Those who benefit from the current system where programs are property offer two arguments in support of their claims to own programs: the emotional argument and the economic argument.

The emotional argument goes like this: “I put my sweat, my heart, my soul into this program. It comes from *me*, it's *mine*!”

This argument does not require serious refutation. The feeling of attachment is one that programmers can cultivate when it suits them; it is not inevitable. Consider, for example, how willingly the same programmers usually sign over all rights to a large corporation for a salary; the emotional attachment mysteriously vanishes.

By contrast, consider the great artists and artisans of medieval times, who didn't even sign their names to their work. To them, the name of the artist was not important. What mattered was that the work was done—and the purpose it would serve. This view prevailed for hundreds of years.

The economic argument goes like this: “I want to get rich (usually described inaccurately as 'making a living'), and if you don't allow me to get rich by programming, then I won't program. Everyone else is like me, so nobody will ever program. And then you'll be stuck with no programs at all!” This threat is usually veiled as friendly advice from the wise.

I'll explain later why this threat is a bluff. First I want to address an implicit assumption that is more visible in another formulation of the argument.

This formulation starts by comparing the social utility of a proprietary program with that of no program, and then concludes that proprietary software development is, on the whole, beneficial, and should be encouraged. The fallacy here is in comparing only two outcomes—proprietary software versus no software—and assuming there are no other possibilities.

Given a system of software copyright, software development is usually linked with the existence of an owner who controls the software's use. As long as this linkage exists, we are often faced with the choice of proprietary software or none. However, this linkage is not inherent or inevitable; it is a consequence of the specific social/legal policy decision that we are questioning: the decision to have owners. To formulate the choice as between proprietary software versus no software is begging the question.

The Argument against Having Owners

The question at hand is, "Should development of software be linked with having owners to restrict the use of it?"

In order to decide this, we have to judge the effect on society of each of those two activities *independently*: the effect of developing the software (regardless of its terms of distribution), and the effect of restricting its use (assuming the software has been developed). If one of these activities is helpful and the other is harmful, we would be better off dropping the linkage and doing only the helpful one.

To put it another way, if restricting the distribution of a program already developed is harmful to society overall, then an ethical software developer will reject the option of doing so.

To determine the effect of restricting sharing, we need to compare the value to society of a restricted (i.e., proprietary) program with that of the same program, available to everyone. This means comparing two possible worlds.

This analysis also addresses the simple counter argument sometimes made that “the benefit to the neighbor of giving him or her a copy of a program is cancelled by the harm done to the owner.” This counter argument assumes that the harm and the benefit are equal in magnitude. The analysis involves comparing these magnitudes, and shows that the benefit is much greater.

To elucidate this argument, let's apply it in another area: road construction.

It would be possible to fund the construction of all roads with tolls. This would entail having toll booths at all street corners. Such a system would provide a great incentive to improve roads. It would also have the virtue of causing the users of any given road to pay for that road. However, a toll booth is an artificial obstruction to smooth driving—artificial, because it is not a consequence of how roads or cars work.

Comparing free roads and toll roads by their usefulness, we find that (all else being equal) roads without toll booths are cheaper to construct, cheaper to run, safer, and more efficient to use. In a poor country, tolls may make the roads unavailable to many citizens. The roads without toll booths thus offer more benefit to society at less cost; they are preferable for society. Therefore, society should choose to fund roads in another way, not by means of toll booths. Use of roads, once built, should be free.

When the advocates of toll booths propose them as *merely* a way of raising funds, they distort the choice that is available. Toll booths do raise funds, but they do something else as well: in effect, they degrade the road. The toll road is not as good as the free road; giving us more or technically superior roads may not be an improvement if this means substituting toll roads for free roads. Of course, the construction of a free road does cost money, which the public must somehow pay. However, this does not imply the inevitability of toll booths. We who must in either case pay will get more value for our money by buying a free road.

I am not saying that a toll road is worse than no road at all. That would be true if the toll were so great that hardly anyone used the road—but this is an unlikely policy for a toll collector. However, as long as the toll booths cause significant waste and inconvenience, it is better to raise the funds in a less obstructive fashion.

To apply the same argument to software development, I will now show that having "toll booths" for useful software programs costs society dearly: it makes the programs more expensive to construct, more expensive to distribute, and less satisfying and efficient to use.

It will follow that program construction should be encouraged in some other way. Then I will go on to explain other methods of encouraging and (to the extent actually necessary) funding software development.

The Harm Done by Obstructing Software

Consider for a moment that a program has been developed, and any necessary payments for its development have been made; now society must choose either to make it proprietary or allow free sharing and use. Assume that the existence of the program and its availability is a desirable thing.

Restrictions on the distribution and modification of the program cannot facilitate its use. They can only interfere. So the effect can only be negative. But how much? And what kind?

Three different levels of material harm come from such obstruction:

- Fewer people use the program.
- None of the users can adapt or fix the program.
- Other developers cannot learn from the program, or base new work on it.

Each level of material harm has a concomitant form of psychosocial harm. This refers to the effect that people's decisions have on their subsequent feelings, attitudes, and predispositions. These changes in people's ways of thinking will then have a further effect on their relationships with their fellow citizens, and can have material consequences.

The three levels of material harm waste part of the value that the program could contribute, but they cannot reduce it to zero. If they waste nearly all the value of the program, then writing the program harms society by at most the effort that went into writing the program. Arguably a program that is profitable to sell must provide some net direct material benefit.

However, taking account of the concomitant psychosocial harm, there is no limit to the harm that proprietary software development can do.

Obstructing Use of Programs

The first level of harm impedes the simple use of a program. A copy of a program has nearly zero marginal cost (and you can pay this cost by doing the work yourself), so in a free market, it would have nearly zero price. A license fee is a significant disincentive to use the

program. If a widely useful program is proprietary, far fewer people will use it.
It is easy to show that the total contribution of a program to society is reduced by assigning an owner to it. Each potential user of the program, faced with the need to pay to use it, may choose to pay, or may forego use of the program. When a user chooses to pay, this is a zero-sum transfer of wealth between two parties. But each time someone chooses to forego use of the program, this harms that person without benefiting anyone. The sum of negative numbers and zeros must be negative.

But this does not reduce the amount of work it takes to *develop* the program. As a result, the efficiency of the whole process, in delivered user satisfaction per hour of work, is reduced.

This reflects a crucial difference between copies of programs and cars, chairs, or sandwiches. There is no copying machine for material objects outside of science fiction. But programs are easy to copy; anyone can produce as many copies as are wanted, with very little effort. This isn't true for material objects because matter is conserved: each new copy has to be built from raw materials in the same way that the first copy was built.

With material objects, a disincentive to use them makes sense, because fewer objects bought means less raw material and work needed to make them. It's true that there is usually also a startup cost, a development cost, which is spread over the production run. But as long as the marginal cost of production is significant, adding a share of the development cost does not make a qualitative difference. And it does not require restrictions on the freedom of ordinary users.

However, imposing a price on something that would otherwise be free is a qualitative change. A centrally imposed fee for software distribution becomes a powerful disincentive.

What's more, central production as now practiced is inefficient even as a means of delivering copies of software. This system involves enclosing physical disks or tapes in superfluous packaging, shipping large numbers of them around the world, and storing them for sale. This cost is presented as an expense of doing business; in truth, it is part of the waste caused by having owners.

Damaging Social Cohesion

Suppose that both you and your neighbor would find it useful to run a certain program. In ethical concern for your neighbor, you should feel that proper handling of the situation will enable both of you to use it. A proposal to permit only one of you to use the program, while restraining the other, is divisive; neither you nor your neighbor should find it acceptable.

Signing a typical software license agreement means betraying your neighbor: “I promise to deprive my neighbor of this program so that I can have a copy for myself.” People who make such choices feel internal psychological pressure to justify them, by downgrading the importance of helping one's neighbors—thus public spirit suffers. This is psychosocial harm associated with the material harm of discouraging use of the program.

Many users unconsciously recognize the wrong of refusing to share, so they decide to ignore the licenses and laws, and share programs anyway. But they often feel guilty about doing so. They know that they must break the laws in order to be good neighbors, but they still consider the laws authoritative, and they conclude that being a good neighbor (which they are) is naughty or shameful. That is also a kind of psychosocial harm, but one can escape it by deciding that these

licenses and laws have no moral force.
Programmers also suffer psychosocial harm knowing that many users will not be allowed to use their work. This leads to an attitude of cynicism or denial. A programmer may describe enthusiastically the work that he finds technically exciting; then when asked, “Will I be permitted to use it?”, his face falls, and he admits the answer is no. To avoid feeling discouraged, he either ignores this fact most of the time or adopts a cynical stance designed to minimize the importance of it.

Since the age of Reagan, the greatest scarcity in the United States is not technical innovation, but rather the willingness to work together for the public good. It makes no sense to encourage the former at the expense of the latter.

Obstructing Custom Adaptation of Programs

The second level of material harm is the inability to adapt programs. The ease of modification of software is one of its great advantages over older technology. But most commercially available software isn't available for modification, even after you buy it. It's available for you to take it or leave it, as a black box—that is all.

A program that you can run consists of a series of numbers whose meaning is obscure. No one, not even a good programmer, can easily change the numbers to make the program do something different.

Programmers normally work with the “source code” for a program, which is written in a programming language such as Fortran or C. It uses names to designate the data being used and the parts of the program, and it represents operations with symbols such as '+' for addition and '-' for subtraction. It is designed to help programmers read and change programs. Here is an example; a program to calculate the distance between two points in a plane:

```
float
 distance (p0, p1)
      struct point p0, p1;
 {
   float xdist = p1.x - p0.x;
   float ydist = p1.y - p0.y;
   return sqrt (xdist * xdist + ydist * ydist);
 }
```

Precisely what that source code means is not the point; the point is that it looks like algebra, and a person who knows this programming language will find it meaningful and clear. By contrast, here is same program in executable form, on the computer I normally used when I wrote this:

```
1314258944    -232267772    -231844864    1634862
1411907592    -231844736    2159150       1420296208
-234880989    -234879837    -234879966    -232295424
1644167167    -3214848      1090581031    1962942495
572518958     -803143692    1314803317
```

Source code is useful (at least potentially) to every user of a program. But most users are not allowed to have copies of the source code. Usually the source code for a proprietary program is kept secret by the owner, lest anybody else learn something from it. Users receive only the files of incomprehensible numbers that the computer will execute. This means that only the program's owner can change the program.

A friend once told me of working as a programmer in a bank for about six months, writing a program similar to something that was commercially available. She believed that if she could have gotten source code for that commercially available program, it could easily have been adapted to their needs. The bank was willing to pay for this, but was not permitted to—the source code was a secret. So she had to do six months of make-work, work that counts in the GNP but was actually waste.

The MIT Artificial Intelligence Lab (AI Lab) received a graphics printer as a gift from Xerox around 1977. It was run by free software to which we added many convenient features. For example, the software would notify a user immediately on completion of a print job. Whenever the printer had trouble, such as a paper jam or running out of paper, the software would immediately notify all users who had print jobs queued. These features facilitated smooth operation.

Later Xerox gave the AI Lab a newer, faster printer, one of the first laser printers. It was driven by proprietary software that ran in a separate dedicated computer, so we couldn't add any of our favorite features. We could arrange to send a notification when a print job was sent to the dedicated computer, but not when the job was actually printed (and the delay was usually considerable). There was no way to find out when the job was actually printed; you could only guess. And no one was informed when there was a paper jam, so the printer often went for an hour without being fixed.

The system programmers at the AI Lab were capable of fixing such problems, probably as capable as the original authors of the program. Xerox was uninterested in fixing them, and chose to prevent us, so we were forced to accept the problems. They were never fixed.

Most good programmers have experienced this frustration. The bank could afford to solve the problem by writing a new program from scratch, but a typical user, no matter how skilled, can only give up. Giving up causes psychosocial harm—to the spirit of self-reliance. It is demoralizing to live in a house that you cannot rearrange to suit your needs. It leads to resignation and discouragement, which can spread to affect other aspects of one's life. People who feel this way are unhappy and do not do good work.

Imagine what it would be like if recipes were hoarded in the same fashion as software. You might say, “How do I change this recipe to take out the salt?” and the great chef would respond, “How dare you insult my recipe, the child of my brain and my palate, by trying to tamper with it? You don't have the judgment to change my recipe and make it work right!”

“But my doctor says I'm not supposed to eat salt! What can I do? Will you take out the salt for me?”

“I would be glad to do that; my fee is only $50,000.” Since the owner has a monopoly on changes, the fee tends to be large.

“However, right now I don't have time. I am busy with a commission to design a new recipe for ship's biscuit for the Navy Department. I might get around to you in about two years.”

Obstructing Software Development

The third level of material harm affects software development. Software development used to be an evolutionary process, where a person would take an existing program and rewrite parts of it for one new feature, and then another person would rewrite parts to add another feature; in some cases, this continued over a period of twenty years. Meanwhile, parts of the program would be “cannibalized” to form the beginnings of other programs.

The existence of owners prevents this kind of evolution, making it necessary to start from scratch when developing a program. It also prevents new practitioners from studying existing programs to learn useful techniques or even how large programs can be structured.

Owners also obstruct education. I have met bright students in computer science who have never seen the source code of a large program. They may be good at writing small programs, but they can't begin to learn the different skills of writing large ones if they can't see how others have done it.

In any intellectual field, one can reach greater heights by standing on the shoulders of others. But that is no longer generally allowed in the software field—you can only stand on the shoulders of the other people *in your own company*.

The associated psychosocial harm affects the spirit of scientific cooperation, which used to be so strong that scientists would cooperate even when their countries were at war. In this spirit, Japanese oceanographers abandoning their lab on an island in the Pacific carefully preserved their work for the invading U.S. Marines, and left a note asking them to take good care of it.

Conflict for profit has destroyed what international conflict spared.

Nowadays scientists in many fields don't publish enough in their papers to enable others to replicate the experiment. They publish only enough to let readers marvel at how much they were able to do. This is certainly true in computer science, where the source code for the programs reported on is usually secret.

It Does Not Matter How Sharing Is Restricted

I have been discussing the effects of preventing people from copying, changing, and building on a program. I have not specified how this obstruction is carried out, because that doesn't affect the conclusion. Whether it is done by copy protection, or copyright, or licenses, or encryption, or ROM cards, or hardware serial numbers, if it *succeeds* in preventing use, it does harm.

Users do consider some of these methods more obnoxious than others. I suggest that the methods most hated are those that accomplish their objective.

Software Should be Free

I have shown how ownership of a program—the power to restrict changing or copying it—is obstructive. Its negative effects are widespread and important. It follows that society shouldn't have owners for programs.

Another way to understand this is that what society needs is free software, and proprietary software is a poor substitute. Encouraging the substitute is not a rational way to get what we need.

Vaclav Havel has advised us to "Work for something because it is good, not just because it stands a chance to succeed." A business making proprietary software stands a chance of success in its own narrow terms, but it is not what is good for society.

Why People Will Develop Software

If we eliminate copyright as a means of encouraging people to develop software, at first less software will be developed, but that software will be more useful. It is not clear whether the overall delivered user satisfaction will be less; but if it is, or if we wish to increase it anyway, there are other ways to encourage development, just as there are ways besides toll booths to raise money for streets. Before I talk about how that can be done, first I want to question how much artificial encouragement is truly necessary.

Programming is Fun

There are some lines of work that few will enter except for money; road construction, for example. There are other fields of study and art in which there is little chance to become rich, which people enter for their fascination or their perceived value to society. Examples include mathematical logic, classical music, and archaeology; and political organizing among working people. People compete, more sadly than bitterly, for the few funded positions available, none of which is funded very well.

They may even pay for the chance to work in the field, if they can afford to.

Such a field can transform itself overnight if it begins to offer the possibility of getting rich. When one worker gets rich, others demand the same opportunity. Soon all may demand large sums of money for doing what they used to do for pleasure. When another couple of years go by, everyone connected with the field will deride the idea that work would be done in the field without large financial returns. They will advise social planners to ensure that these returns are possible, prescribing special privileges, powers, and monopolies as necessary to do so.

This change happened in the field of computer programming in the 1980s. In the 1970s, there were articles on "computer addiction": users were "onlining" and had hundred-dollar-a-week habits. It was generally understood that people frequently loved programming enough to break up their marriages.

Today, it is generally understood that no one would program except for a high rate of pay. People have forgotten what they knew back then. When it is true at a given time that most people will work in a certain field only for high pay, it need not remain true. The dynamic of change can run in reverse, if society provides an impetus. If we take away the possibility of great wealth, then after a while, when the people have readjusted their attitudes, they will once again be eager to work in the field for the joy of accomplishment.

The question “How can we pay programmers?” becomes an easier question when we realize that it's not a matter of paying them a fortune. A mere living is easier to raise.

Funding Free Software

Institutions that pay programmers do not have to be software houses. Many other institutions already exist that can do this.

Hardware manufacturers find it essential to support software development even if they cannot control the use of the software. In 1970, much of their software was free because they did not consider restricting it. Today, their increasing willingness to join consortiums shows their realization that owning the software is not what is really important for them.

Universities conduct many programming projects. Today they often sell the results, but in the 1970s they did not. Is there any doubt that universities would develop free software if they were not allowed to sell software? These projects could be supported by the same government contracts and grants that now support proprietary software development.

It is common today for university researchers to get grants to develop a system, develop it nearly to the point of completion and call that “finished”, and then start companies where they really finish the project and make it usable. Sometimes they declare the unfinished version “free”; if they are thoroughly corrupt, they instead get an exclusive license from the university. This is not a secret; it is openly admitted by everyone concerned. Yet if the researchers were not exposed to the temptation to do these things, they would still do their research.

Programmers writing free software can make their living by selling services related to the software. I have been hired to port the GNU C compiler to new hardware, and to make user-interface extensions to GNU Emacs. (I offer these improvements to the public once they are done.) I also teach classes for which I am paid.

I am not alone in working this way; there is now a successful, growing corporation which does no other kind of work. Several other companies also provide commercial support for the free software of the GNU system. This is the beginning of the independent software support industry—an industry that could become quite large if free software becomes prevalent. It provides users with an option generally unavailable for proprietary software, except to the very wealthy.

New institutions such as the Free Software Foundation can also fund programmers. Most of the Foundation's funds come from users buying tapes through the mail. The software on the tapes is free, which means that every user has the freedom to copy it and change it, but many nonetheless pay to get copies. (Recall that “free software” refers to freedom, not to price.) Some users who already have a copy order tapes as a way of making a contribution they feel we deserve. The Foundation also receives sizable donations from computer manufacturers.

The Free Software Foundation is a charity, and its income is spent on hiring as many programmers as possible. If it had been set up as a business, distributing the same free software to the public for the same fee, it would now provide a very good living for its founder.

Because the Foundation is a charity, programmers often work for the Foundation for half of what they could make elsewhere. They do this because we are free of bureaucracy, and because they feel satisfaction in knowing that their work will not be obstructed from use. Most of all, they do it because programming is fun. In addition, volunteers have written many useful programs for us. (Even technical writers have begun to volunteer.)

This confirms that programming is among the most fascinating of all fields, along with music and art. We don't have to fear that no one will want to program.

What Do Users Owe to Developers?

There is a good reason for users of software to feel a moral obligation to contribute to its support. Developers of free software are contributing to the users' activities, and it is both fair and in the long-term interest of the users to give them funds to continue.

However, this does not apply to proprietary software developers, since obstructionism deserves a punishment rather than a reward.

We thus have a paradox: the developer of useful software is entitled to the support of the users, but any attempt to turn this moral obligation into a requirement destroys the basis for the obligation. A developer can either deserve a reward or demand it, but not both.

I believe that an ethical developer faced with this paradox must act so as to deserve the reward, but should also entreat the users for voluntary donations. Eventually the users will learn to support developers without coercion, just as they have learned to support public radio and television stations.

What Is Software Productivity?

If software were free, there would still be programmers, but perhaps fewer of them. Would this be bad for society?

Not necessarily. Today the advanced nations have fewer farmers than in 1900, but we do not think this is bad for society, because the few deliver more food to the consumers than the many used to do. We call this improved productivity.

Free software would require far fewer programmers to satisfy the demand, because of increased software productivity at all levels:

- Wider use of each program that is developed.
- The ability to adapt existing programs for customization instead of starting from scratch.
- Better education of programmers.
- The elimination of duplicate development effort.

Those who object to cooperation claiming it would result in the employment of fewer programmers are actually objecting to increased productivity. Yet these people usually accept the widely held belief that the software industry needs increased productivity. How is this?

“Software productivity” can mean two different things: the overall productivity of all software development, or the productivity of individual projects. Overall productivity is what society would like to improve, and the most straightforward way to do this is to eliminate the artificial obstacles to cooperation which reduce it. But researchers who study the field of "software productivity" focus only on the second, limited, sense of the term, where improvement requires difficult technological advances.

Is Competition Inevitable?

Is it inevitable that people will try to compete, to surpass their rivals in society? Perhaps it is. But competition itself is not harmful; the harmful thing is *combat*.

There are many ways to compete. Competition can consist of trying to achieve ever more, to outdo what others have done. For example, in the old days, there was competition among programming wizards—competition for who could make the computer do the most amazing thing, or for who could make the shortest or fastest program for a given task. This kind of competition can benefit everyone, *as long as* the spirit of good sportsmanship is maintained.

Constructive competition is enough competition to motivate people to great efforts. A number of people are competing to be the first to have visited all the countries on Earth; some even spend fortunes trying to do this. But they do not bribe ship captains to strand their rivals on desert islands. They are content to let the best person win.

Competition becomes combat when the competitors begin trying to impede each other instead of advancing themselves—when "Let the best person win" gives way to "Let me win, best or not." Proprietary software is harmful, not because it is a form of competition, but because it is a form of combat among the citizens of our society.

Competition in business is not necessarily combat. For example, when two grocery stores compete, their entire effort is to improve their own operations, not to sabotage the rival. But this does not demonstrate a special commitment to business ethics; rather, there is little scope for combat in this line of business short of physical violence. Not all areas of business share this characteristic. Withholding information that could help everyone advance is a form of combat.

Business ideology does not prepare people to resist the temptation to combat the competition. Some forms of combat have been banned with antitrust laws, truth in advertising laws, and so on, but rather than generalizing this to a principled rejection of combat in general, executives invent other forms of combat which are not specifically prohibited. Society's resources are squandered on the economic equivalent of factional civil war.

Conclusion

We like to think that our society encourages helping your neighbor; but each time we reward someone for obstructionism, or admire them for the wealth they have gained in this way, we are sending the opposite message.

Software hoarding is one form of our general willingness to disregard the welfare of society for personal gain. We can trace this disregard from Ronald Reagan to Dick Cheney, from Exxon to Enron, from failing banks to failing schools. We can measure it with the size of the homeless population and the prison population. The antisocial spirit feeds on itself, because the more we see that other people will not help us, the more it seems futile to help them. Thus society decays into a jungle.

If we don't want to live in a jungle, we must change our attitudes. We must start sending the message that a good citizen is one who cooperates when appropriate, not one who is successful at taking from others. I hope that the free software movement will contribute to this: at least in one area, we will replace the jungle with a more efficient system which encourages and runs on voluntary cooperation.

Footnotes

1. The word "free" in "free software" refers to freedom, not to price; the price paid for a copy of a free program may be zero, or small, or (rarely) quite large.

2. The issues of pollution and traffic congestion do not alter this conclusion. If we wish to make driving more expensive to discourage driving in general, it is disadvantageous to do this using toll booths, which contribute to both pollution and congestion. A tax on gasoline is much better. Likewise, a desire to enhance safety by limiting maximum speed is not relevant; a free-access road enhances the average speed by avoiding stops and delays, for any given speed limit.

3. One might regard a particular computer program as a harmful thing that should not be available at all, like the Lotus Marketplace database of personal information, which was withdrawn from sale due to public disapproval. Most of what I say does not apply to this case, but it makes little sense to argue for having an owner on the grounds that the owner will make the program less available. The owner will not make it *completely* unavailable, as one would wish in the case of a program whose use is considered destructive.

* * *

Exercise :

A) How propriety software are damaging our economy.

B) How you can do the business by using free open source software

C) How the funding help is available to free software

D) What do you mean by software productivity?

E) How the philosophy of free software would benefit Indian ecosystem?

6. Free Software and Education

How Does Free Software Relate to Education?

Software freedom plays a fundamental role in education. Educational institutions of all levels should use and teach Free Software because it is the only software that allows them to accomplish their essential missions: to disseminate human knowledge and to prepare students to be good members of their community. The source code and the methods of Free Software are part of human knowledge. On the contrary, proprietary software is secret, restricted knowledge, which is the opposite of the mission of educational institutions. Free Software supports education, proprietary software forbids education.

Free Software is not just a technical question; it is an ethical, social, and political question. It is a question of the human rights that the users of software ought to have. Freedom and cooperation are essential values of Free Software. The GNU System implements these values and the principle of sharing, since sharing is good and beneficial to human progress.

Why Educational Institutions Should Use and Teach Free Software - by www.gnu.org

"Schools should teach their students to be citizens of a strong, capable, independent and free society."

These are the main reasons why universities and schools of all levels should use exclusively Free Software.

Sharing

Schools should teach the value of sharing by setting an example. Free software supports education by allowing the sharing of knowledge and tools:

- **Knowledge**. Many young students have a talent for programing; they are fascinated with computers and eager to learn how their systems work. With proprietary software, this information is a secret so teachers have no way of making it available to their students. But if it is Free Software, the teacher can explain the basic subject and then hand out the source code for the student to read and learn.
- **Tools**. Teachers can hand out to students copies of the programs they use in the classroom so that they can use them at home. With Free Software, copying is not only authorized, it is encouraged.

Social Responsibility

Computing has become an essential part of everyday life. Digital technology is transforming society very quickly, and schools have an influence on the future of society. Their mission is to get students ready to participate in a free digital society by teaching them the skills to make it easy for them to take control of their own lives.

Software should not be under the power of a software developer who unilaterally makes decisions that nobody else can change. Educational institutions should not allow proprietary software companies to impose their power on the rest of society and its future.

Independence

Schools have an ethical responsibility to teach strength, not dependency on a single product or a specific powerful company. Furthermore, by choosing to use Free Software, the school itself gains independence from any commercial interests and it avoids vendor lock-in.

- Proprietary software companies use schools and universities as a springboard to reach users and thus impose their software on society as a whole. They offer discounts, or even gratis copies of their proprietary programs to educational institutions, so that students will learn to use them and become dependent on them. After these students graduate, neither they nor their future employers will be offered discounted copies. Essentially, what these companies are doing is they are recruiting schools and universities into agents to lead people to permanent lifelong dependency.
- Free software licenses do not expire, which means that once Free Software is adopted, institutions remain independent from the vendor. Moreover, Free Software licenses grant users the rights not only to use the software as they wish, to copy it and distribute it, but also to modify it in order to meet their own needs. Therefore, if institutions eventually wish to implement a particular function in a piece of software, they can engage the services of any developer to accomplish the task, independently from the original vendor.

Learning

When deciding where they will study, more and more students are considering whether a university teaches computer science and software development using Free Software. Free software means that students are free to study how the programs work and to learn how to adapt them for their own needs. Learning about Free Software also helps in studying software development ethics and professional practice.

Saving

This is an obvious advantage that will appeal immediately to many school administrators, but it is a marginal benefit. The main point of this aspect is that by being authorized to distribute copies of the programs at little or no cost, schools can actually aid families facing financial issues, thus promoting fairness and equal opportunities of learning among students.

Quality

Stable, secure and easily installed Free Software solutions are available for education already. In any case, excellence of performance is a secondary benefit; the ultimate goal is freedom for computer users.

* * *

7. Why Schools Should Exclusively Use Free Software - by Richard Stallman

Educational activities, including schools of all levels from kindergarten to university, have a moral duty to teach only free software.

All computer users ought to insist on free software: it gives users the freedom to control their own computers—with proprietary software, the program does what its owner or developer wants it to do, not what the user wants it to do.

Free software also gives users the freedom to cooperate with each other, to lead an upright life. These reasons apply to schools as they do to everyone. However, the purpose of this article is to present the additional reasons that apply specifically to education.

Free software can save schools money, but this is a secondary benefit. Savings are possible because free software gives schools, like other users, the freedom to copy and redistribute the software; the school system can give a copy to every school, and each school can install the program in all its computers, with no obligation to pay for doing so.

This benefit is useful, but we firmly refuse to give it first place, because it is shallow compared to the important ethical issues at stake. Moving schools to free software is more than a way to make education a little “better”: it is a matter of doing good education instead of bad education. So let's consider the deeper issues.

Schools have a social mission: to teach students to be citizens of a strong, capable, independent, cooperating and free society. They should promote the use of free software just as they promote conservation and voting. By teaching students free software, they can graduate citizens ready to live in a free digital society. This will help society as a whole escape from being dominated by megacorporations.

In contrast, to teach a nonfree program is implanting dependence, which goes counter to the schools' social mission. Schools should never do this.

Why, after all, do some proprietary software developers offer gratis copies(1) of their nonfree programs to schools? Because they want to use the schools to implant dependence on their products, like tobacco companies distributing gratis cigarettes to school children(2). They will not give gratis copies to these students once they've graduated, nor to the companies that they go to work for. Once you're dependent, you're expected to pay, and future upgrades may be expensive.

Free software permits students to learn how software works. Some students, natural-born programmers, on reaching their teens yearn to learn everything there is to know about their computer and its software. They are intensely curious to read the source code of the programs that they use every day.

Proprietary software rejects their thirst for knowledge: it says, “The knowledge you want is a secret—learning is forbidden!" Proprietary software is the enemy of the spirit of education, so it should not be tolerated in a school, except as an object for reverse engineering.

Free software encourages everyone to learn. The free software community rejects the "priesthood of technology", which keeps the general public in ignorance of how technology works; we encourage students of any age and situation to read the source code and learn as much as they want to know.

Schools that use free software will enable gifted programming students to advance. How do natural-born programmers learn to be good programmers? They need to read and understand real programs that people really use. You learn to write good, clear code by reading lots of code and writing lots of code. Only free software permits this.

How do you learn to write code for large programs? You do that by writing lots of changes in existing large programs. Free Software lets you do this; proprietary software forbids this. Any school can offer its students the chance to master the craft of programming, but only if it is a free software school.

The deepest reason for using free software in schools is for moral education. We expect schools to teach students basic facts and useful skills, but that is only part of their job. The most fundamental task of schools is to teach good citizenship, including the habit of helping others. In the area of computing, this means teaching people to share software. Schools, starting from nursery school, should tell their students, "If you bring software to school, you must share it with the other students. You must show the source code to the class, in case someone wants to learn. Therefore bringing nonfree software to class is not permitted, unless it is for reverse-engineering work."

Of course, the school must practice what it preaches: it should bring only free software to class (except objects for reverse-engineering), and share copies including source code with the students so they can copy it, take it home, and redistribute it further.
Teaching the students to use free software, and to participate in the free software community, is a hands-on civics lesson. It also teaches students the role model of public service rather than that of tycoons. All levels of school should use free software.

If you have a relationship with a school —if you are a student, a teacher, an employee, an administrator, a donor, or a parent— it's your responsibility to campaign for the school to migrate to free software. If a private request doesn't achieve the goal, raise the issue publicly in those communities; that is the way to make more people aware of the issue and find allies for the campaign.

* * *

Exercise : A) Find out the free or open source alternative of the most common proprietary software we are in our daily life.

B) Calculate the expenses which may our on each machine after installing the most essential (common) proprietary software and then multiply the amount with no. of machines available in your school.

8. THE GNU License

The **GNU General Public License** (**GNU GPL** or **GPL**) is a widely used free software license, which guarantees end users the freedom to run, study, share and modify the software. The license was originally written by Richard Stallman of the Free Software Foundation (FSF) for the GNU Project, and grants the recipients of a computer program the rights of the Free Software Definition.

The GPL is a copyleft license, which means that derivative work can only be distributed under the same license terms. This is in distinction to permissive free software licenses, of which the BSD licenses and the MIT License are widely used examples. GPL was the first copyleft license for general use.

The GPL was written by Richard Stallman in 1989, for use with programs released as part of the GNU project.

* * *

9. The Open Source Philosophy

as mentioned on https://opensource.org

What is "Open Source" software?

Generally, Open Source software is software that can be freely accessed, used, changed, and shared (in modified or unmodified form) by anyone. Open source software is made by many people, and distributed under licenses that comply with the Open Source Definition.

The internationally recognized Open Source Definition provides ten criteria that must be met for any software license, and the software distributed under that license, to be labeled "Open Source software." Only software licensed under an OSI-approved Open Source license should be labeled "Open Source" software.

Using Open Source for Commercial Purpose

All Open Source software can be used for commercial purpose; the Open Source Definition guarantees this. You can even sell Open Source software.

However, note that commercial is not the same as proprietary. If you receive software under an Open Source license, you can always use that software for commercial purposes, but that doesn't always mean you can place further restrictions on people who receive the software from you. In particular, copyleft-style Open Source licenses require that, in at least some cases, when you distribute the software, you must do so under the same license you received it under.

10. The Copyleft Theory

"Copyleft" refers to licenses that allow derivative works but require them to use the same license as the original work. For example, if you write some software and release it under the GNU General Public License (a widely-used copyleft license), and then someone else modifies that software and distributes their modified version, the modified version must be licensed under the GNU GPL too — including any new code written specifically to go into the modified version.

Both the original and the new work are Open Source; the copyleft license simply ensures that property is perpetuated to all downstream derivatives. (There is at least one copyleft license, the Affero GPL, that even requires you to offer the source code, under the AGPL, to anyone to whom you make the software's functionality available as a network service — however, most copyleft licenses activate their share-and-share-alike requirement on distribution of a copy of the software itself. You should read the license to understand its requirements for source code distribution.)

Most copyleft licenses are Open Source, but not all Open Source licenses are copyleft. When an Open Source license is not copyleft, that means software released under that license can be used as part of programs distributed under other licenses, including proprietary (non-open-source) licenses.

For example, the BSD license is a non-copyleft Open Source license. Such licenses are usually called either "non-copyleft" or "permissive" open source licenses Copyleft provisions apply only to actual derivatives, that is, cases where an existing copylefted work was modified. Merely distributing a copyleft work alongside a non-copyleft work does not cause the latter to fall under the copyleft terms.

* * *

11. List of Free / FOSS Software for Education Educational suites

- ATutor — a web-based Learning Content Management System (LCMS)
- Chamilo — a web-based e-learning and content management system
- Claroline — a collaborative Learning Management System
- DoceboLMS
- eFront — an icon-based learning management system
- FlightPath — academic advising software for universities
- GCompris
- Gnaural — Brainwave entrainment software
- IUP Portfolio
- ILIAS — a web-based learning management system (LMS)
- Moodle — a free and open-source learning management system
- OLAT — a web-based Learning Content Management System
- Omeka
- openSIS — a web-based Student Information and School Management system
- Sakai Project — a web-based learning management system
- SWAD – a web-based learning management system
- Tux Paint — a paint application for 3–12 year olds

Geography

- KGeography

Learning support

Main category: Free learning support software

Language

- Kiten
- KVerbos

Typing

- KTouch
- Tux Typing

Other educational programs

KEduca

GMIP

* * *

12. Linux an ultimate hero

(Note : The below mentioned text is a note on Linux mentioned on the official site https://www.linux.com/what-is-linux, Students are requested to browse the site,in case they want more information on Linux)

What is Linux?

From smartphones to cars, supercomputers and home appliances, the Linux operating system is everywhere.

Linux. It's been around since the mid '90s, and has since reached a user-base that spans industries and continents. For those in the know, you understand that Linux is actually everywhere.

It's in your phones, in your cars, in your refrigerators, your Roku devices. It runs most of the Internet, the supercomputers making scientific breakthroughs, and the world\'s stock exchanges. But before Linux became the platform to run desktops, servers, and embedded systems across the globe, it was (and still is) one of the most reliable, secure, and worry-free operating systems available.

For those not in the know, worry not – here is all the information you need to get up to speed on the Linux platform.

What is Linux?
Just like Windows XP, Windows 7, Windows 8, and Mac OS X, Linux is an operating system. An operating system is software that manages all of the hardware resources associated with your desktop or laptop. To put it simply – the operating system manages the communication between your software and your hardware. Without the operating system (often referred to as the "OS"), the software wouldn't function.

The OS is comprised of a number of pieces:

- **The Bootloader:** The software that manages the boot process of your computer. For most users, this will simply be a splash screen that pops up and eventually goes away to boot into the operating system.
- **The kernel:** This is the one piece of the whole that is actually called "Linux". The kernel is the core of the system and manages the CPU, memory, and peripheral devices. The kernel is the "lowest" level of the OS.
- **Daemons:** These are background services (printing, sound, scheduling, etc) that either startup during boot, or after you log into the desktop.
- **The Shell:** You've probably heard mention of the Linux command line. This is the shell – a command process that allows you to control the computer via commands typed into a text interface. This is what, at one time, scared people away from Linux the most (assuming they had to learn a seemingly archaic command line structure to make Linux work). This is no longer the case. With modern desktop Linux, there is no

need to ever touch the command line.

- **Graphical Server:** This is the sub-system that displays the graphics on your monitor. It is commonly referred to as the X server or just "X".

- **Desktop Environment:** This is the piece of the puzzle that the users actually interact with. There are many desktop environments to choose from (Unity, GNOME, Cinnamon, Enlightenment, KDE, XFCE, etc). Each desktop environment includes built-in applications (such as file managers, configuration tools, web browsers, games, etc).

- **Applications:** Desktop environments do not offer the full array of apps. Just like Windows and Mac, Linux offers thousands upon thousands of high-quality software titles that can be easily found and installed. Most modern Linux distributions (more on this in a moment) include App Store-like tools that centralize and simplify application installation. For example: Ubuntu Linux has the Ubuntu Software Center which allows you to quickly search among the thousands of apps and install them from one centralized location.

Why use Linux?

This is the one question that most people ask. Why bother learning a completely different computing environment, when the operating system that ships with most desktops, laptops, and servers works just fine? To answer that question, I would pose another question. Does that operating system you're currently using *really* work “just fine”? Or are you constantly battling viruses, malware, slowdowns, crashes, costly repairs, and licensing fees?

If you struggle with the above, and want to free yourself from the constant fear of losing data or having to take your computer in for the “yearly clean up,” Linux might be the perfect platform for you. Linux has evolved into one of the most reliable computer ecosystems on the planet. Combine that reliability with zero cost of entry and you have the perfect solution for a desktop platform.

That's right, zero cost of entry...as in free. You can install Linux on as many computers as you like without paying a cent for software or server licensing (including costly Microsoft Client Access License – CALs).

Let's take a look at the cost of a Linux server, in comparison to Windows Server 2012. The price of the Windows Server 2012 software alone can run up to $1,200.00 USD. That doesn't include CALs, and licenses for other software you may need to run (such as a database, a web server, mail server, etc). With the Linux server...it's all free and easy to install. In fact, installing a full blown web server (that includes a database server), is just a few clicks or commands away (take a look at “Easy LAMP Server Installation” to get an idea how simple it can be).

If you're a system administrator, working with Linux is a dream come true. No more daily babysitting servers. In fact, Linux is as close to “set it and forget it” as you will ever find. And, on the off chance, one service on the server requires restarting, re-configuring, upgrading, etc...most likely the rest of the server won't be affected.

Be it the desktop or a server, if zero cost isn't enough to win you over – what about having an operating system that will work, trouble free, for as long as you use it? I've personally used Linux for nearly twenty years (as a desktop and server platform) and have not once had an issue with malware, viruses, or random computer slow-downs. It's *that* stable. And server reboots? Only if the kernel is updated. It is not out of the ordinary for a Linux server to go years without being rebooted. That's stability and dependability.

Linux is also distributed under an open source license. Open source follows the following key philosophies:

- The freedom to run the program, for any purpose.
- The freedom to study how the program works, and change it to make it do what you wish.
- The freedom to redistribute copies so you can help your neighbor.
- The freedom to distribute copies of your modified versions to others.
- The above are crucial to understanding the community that comes together to create the Linux platform. It is, without a doubt, an operating system that is "by the people, for the people". These philosophies are also one of the main reasons a large percentage of people use Linux. It's about freedom and freedom of choice.

What is a "distribution?"

Linux has a number of different versions to suit nearly any type of user. From new users to hard-core users, you'll find a “flavor” of Linux to match your needs. These versions are called distributions (or, in the short form, “distros.”) Nearly every distribution of Linux can be downloaded for free, burned onto disk (or USB thumb drive), and installed (on as many machines as you like).

The most popular Linux distributions are:

- Ubuntu Linux
- Linux Mint
- Arch Linux
- Deepin
- Fedora
- Debian
- openSUSE.

Each distribution has a different take on the desktop. Some opt for very modern user interfaces (such as Ubuntu's Unity, above, and Deepin's Deepin Desktop), whereas others stick with a more traditional desktop environment (openSUSE uses KDE).

More Resources

If you're looking for one of the most reliable, secure, and dependable platforms for both the desktop and the server, look no further than one of the many Linux distributions. With Linux you can assure your desktops will be free of trouble, your servers up, and your support requests at a minimum.

If you're looking for more resources to help guide you through your lifetime with Linux, check out the following resources:

- Linux.com: Everything you need to know about Linux (news, how-tos, answers, forums, and more)
- Linux.org: Everything about the Linux kernel (with plenty of beginner, intermediate, and advanced tutorials)
- Howtoforge: Linux tutorials

- Linux Documentation Project: Plenty of documentation (some may be out of date)
- Linux Knowledge Base and Tutorial: Plenty of tutorials.

-- Source Curtsy - The Linux Foundation - https://www.linux.com/what-is-linux

The Wikipedia List of Linux adopters

Wikipedia shares a huge list of Linux adopters worldwide out of them some prominent adopters are as follows.

Asia

The People's Republic of China exclusively uses Linux as the operating system for its Loongson processor family, with the aim of technology independence.

Kylin, used by People's Liberation Army in The People's Republic of China. The first version used FreeBSD, but since release 3.0, it employs Linux.

State owned Industrial and Commercial Bank of China (ICBC) is installing Linux in all of its 20,000 retail branches as the basis for its web server and a new terminal platform. (2005)

The Government of Kerala, India, announced its official support for free/open-source software in its State IT Policy of 2001, which was formulated after the first-ever free software conference in India, "Freedom First!", held in July 2001 in Trivandrum, the capital of Kerala, where Richard Stallman inaugurated the Free Software

Foundation of India. Since then, Kerala's IT Policy has been significantly influenced by FOSS, with several major initiatives such as IT@School Project, possibly the largest single-purpose deployment of Linux in the world, and leading to the formation of the International Centre for Free and Open Source Software (ICFOSS) in 2009.

In March 2014, with the end of support for Windows XP, the Government of Tamil Nadu has advised all its departments to install BOSS Linux (Bharat Operating System Solutions).

USA

In July 2001,the White House started switching their web servers to an operating system based on Red Hat Linux and using the Apache HTTP Server.The installation was completed in February 2009.In October 2009, the White House servers adopted Drupal, an open source content management system software distribution.

The United States Department of Defense uses Linux - "the U.S. Army is “the” single largest install base for Red Hat Linux] and the US Navy nuclear submarine fleet runs on Linux,including their sonar systems.

In June 2012, the US Navy signed a US$27,883,883 contract with Raytheon to install Linux ground control software for its fleet of vertical take-off and landing (VTOL) Northrup-Grumman MQ8B Fire Scout drones. The contract involves Naval Air Station Patuxent

River, Maryland, which has already spent $5,175,075 in preparation for the Linux systems.

In April 2006, the US Federal Aviation Administration announced that it had completed a migration to Red Hat Enterprise Linux in one third of the scheduled time and about 15 million dollars under budget. The switch saved a further $15 million in datacenter operating costs.
The US National Nuclear Security Administration operates the world's tenth fastest supercomputer, the IBM Roadrunner, which uses Red Hat Enterprise Linux along with Fedora as its operating systems.

The city government of Largo, Florida, USA uses Linux and has won international recognition for their implementation, indicating that it provides "extensive savings over more traditional alternatives in city-wide applications."

U.K.

In 2013, Westcliff High School for Girls in the United Kingdom successfully moved from Windows to OpenSUSE Linux.
Orwell High School, in Felixstowe, England, school with about 1,000 students, has switched to Linux. The school has just received Specialist School for Technology status through a government initiative.

Switzerland

All primary and secondary public schools in the Swiss Canton of Geneva, have switched to using Ubuntu for the PCs used by teachers and students in 2013-14. The switch has been completed by all of the

170 primary public schools and over 2,000 computers.

Americas

Brazil has 35 million students in over 50,000 schools using 523,400 computer stations all running Linux.

22,000 students in the US state of Indiana had access to Linux Workstations at their high schools in 2006.

In 2009, Venezuela's Ministry of Education began a project called Canaima-educativo, to provide all students in public schools with "Canaimita" laptop computers with the Canaima Debian-based Linux distribution pre-installed, as well as with open source educational content

India

- The Indian government's tablet computer initiative for student use employs Linux as the operating system as part of its drive to produce a tablet PC for under 1,500 rupees (US$35).

- The Indian state of Tamil Nadu plans to distribute 100,000 Linux laptops to its students.

- Government officials of Kerala, India announced they will use only free software, running on the Linux platform, for computer education, starting with the 2,650 government and government-aided high schools.

- The Indian state of Tamil Nadu has issued a directive to local government departments asking them to switch over to open source software, in the wake of Microsoft's decision to end support for Windows XP in April 2014

Note : Full list can be referred at Wikipedia under linux Adopters page.

* * *

13. Bharat Operating System Solutions

Bharat Operating System Solutions (BOSS Linux) is a free and open source Linux distribution developed by the National Resource Centre for Free/Open Source Software (NRCFOSS) of India. The latest stable release, version 6.0, was released in March 2015.

This software package has been described as "India's own PC operating system" and "the most meaningful product to come out of the Indian software industry in decades — and let's recognise it, this is work that a government department had to do." The software has also been endorsed by the Government of India for adoption and implementation on a national scale. It was developed at Centre for Development of Advanced Computing (CDAC), Chennai INDIA. BOSS GNU/Linux is an "LSB certified" Linux distribution: the software has been certified by the Linux Foundation for compliance with the Linux Standard Base standard.BOSS GNU/Linux is derived from Debian GNU/Linux.

BOSS (Bharat Operating System Solutions) GNU/Linux distribution developed by C-DAC (Centre for Development of Advanced Computing) derived from Debian for enhancing the use of Free/

Open source software throughout India. BOSSGNU/Linux – a key deliverable of NRCFOSS has upgraded from entry-level server to advanced server

Official Website : https://www.bosslinux.in/
(Content Source : Wikipedia and Bosslinux)

* * *

Project: Install the boss linux operating system on your machine under the supervision of computer teacher compare the feature of BOSS linux with other proprietary operating system and write down your observations.

14. Apache Openoffice : A free office publishing tool for all

Apache OpenOffice is the leading open-source office software suite for word processing, spreadsheets, presentations, graphics, databases and more. It is available in many languages and works on all common computers. It stores all your data in an international open standard format and can also read and write files from other common office software packages. It can be downloaded and used completely free of charge for any purpose

Some super exciting advantages of Apache OpenOffice

Great software

Apache OpenOffice is the result of over twenty years' software engineering. Designed from the start as a single piece of software, it has a consistency other products cannot match. A completely open development process means that anyone can report bugs, request new features, or enhance the software. The result: Apache OpenOffice does everything you want your office software to do, the way you want it to.

Easy to use

Apache OpenOffice is easy to learn, and if you're already using another office software package, you'll take to OpenOffice straight away. Our world-wide native-language community means that OpenOffice is probably available and supported in your own language. And if you already have files from another office package - OpenOffice will probably read them with no difficulty.

and it's free

Best of all, Apache OpenOffice can be downloaded and used entirely **free** of any license fees. Like all Apache Software Foundation software, Apache OpenOffice is free to use. Apache OpenOffice is released under the Apache 2.0 License. This means you may use it for any purpose - domestic, commercial, educational, public administration. You may install it on as many computers as you like. You may make copies and give them away to family, friends, students, employees - anyone you like.

Why Apache OpenOffice: Education

Education establishments of all levels (primary, secondary, college, university...) find Apache OpenOffice meets the needs of both teachers and students. The flexible word processor, powerful spreadsheet, dynamic graphics, database access and more meet all requirements for an office software package.

With an open-source license, OpenOffice can be freely used and distributed with no license worries.

- For pupils and students
- Apache OpenOffice forms an ideal teaching platform for core computer literacy skills, without tying students to commercial products. The free software license means students can be given copies of software to use at home - perfectly legally - a useful 'added value'. For IT students,

OpenOffice's component based software is also an ideal platform for developing IT skills and understanding real-life software engineering.

- For teachers and academics
- Apache OpenOffice is also an ideal platform for creating teaching materials and managing administrative tasks. For example, the Writer word processor is easy to use for simple memos, but also powerful enough to cope with complex dissertations. For IT staff, the open-source software license means an end to license compliance worries and the threat of software audits. OpenOffice is developed, translated, and supported by an international community linked by the internet, opening exciting possibilities for school projects.
- Open for all
- Apache OpenOffice is a leading international force in the movement for digital inclusion - making software of the highest quality available to all, regardless of income. OpenOffice is available in a wide variety of languages, and we actively encourage local teams to produce versions for local languages. We develop software on an open-source process - the computing equivalent of peer-reviewed publishing - creating software of the highest quality.

(Content Source : https://www.openoffice.org/why/why_edu.html **, Please visit the website for more information.)**

* * *

Project: Install Apache open office in a machine under the supervision of your computer teacher compare the features of open office with other office publishing tools and make a note.

MCE Society's
P. A. Inamdar College of Visual Effects, Design & Art
VEDA
www.veda-edu.com

Global Certifications

unity Certification

CERTIPORT®
A PEARSON VUE BUSINESS
AUTHORISED TESTING CENTER

Microsoft
Office Specialist

intuit quickbooks® Online
Certified User

ESB
Entrepreneurship and Small Business

IC3
DIGITAL LITERACY
CERTIFICATION

AUTODESK®
Certified User

Adobe®
CERTIFIED ASSOCIATE

EC-Council
Associate

Do it from anywhere - anytime
As per your convenience

+91 888 880 8544/ 855 096 6911

www.itolympiad.in

CiTiS
knowledge integrated

CITIS Infotech : Knowledge Partner