Following are the Solution Notes and Marking Scheme for the 20 mark question on Professional Practice and Ethics:

The Question - with parts lettered for subsequent reference:

(A) What kinds of law can be used in protecting intellectual property? [6 marks]
(B) What problems are faced with each of these kinds of law in protecting software as a kind of intellectual property (i.e. creating proprietary software)? [4 marks]
(C) What is the ethical justification for protection of software as a kind of intellectual property? [3 marks]
(D) What arguments are used to oppose this justification? [3 marks]
(E) What alternative is there to proprietary software? [2 marks]
(F) How could computer professionals support themselves without proprietary software? [2 marks]

(A) & (B)

For the first part of the question there are three kinds of law which were discussed in lectures and also in the printed notes that were handed out. Along with the discussion of copyright law and the law of trade secrets was a discussion of the difficulties found in using that kind of law to protect software as intellectual property - these provide the answer to the second part of the question. The notes cover the difficulties with copyright and trade secrets - the difficulties of patent law were only discussed in lectures. For part (A) of the question give a mark for each kind of law correctly mentioned and a further mark if anything sensible is said about its purpose and/or how it works. For part (B) of the question give a mark if any real difficulty is mentioned and another mark if it is a part of a reasonable discussion.

1. Copyright

For copyright purposes a computer programme is defined as "a set of statements or instructions used directly or indirectly in a computer in order to bring about a certain result." By referring to instructions used "directly or indirectly" the definition includes both the source programme and the object programme. What the definition does not include is the idea behind the programme, the algorithm. The law of copyright applies to the *expression* of the idea, not to the idea itself. The main difficulty in the use of copyright law to protect software property rights is in the more general aspects of programmes, their "structure, sequence and organisation", and especially in the user interfaces of programmes, their "look and feel". In these aspects the underlying code can be quite different but still give rise to the same look and feel.

2. Trade secrets

Trade secrecy laws generally give a company the right to keep certain kinds of information secret. The laws are aimed at protecting companies from having competitors find out what it is about their products that give them an advantage in the marketplace. The social utility of such laws is that they encourage improvement of products or production techniques by establishing the rights of business to keep secret the method of those improvements. In general, for a company to claim that some information constitutes a trade secret, it must satisfy four requirements:

(1) The information must be novel in some way;
(2) It must represent an economic investment
(3) It must have involved some effort to develop
(4) The company must make an effort to keep it secret.

Software typically satisfies the first three requirements, but the fourth requirement is sometimes more difficult - particularly when the software is sold. With software that is kept "in-house" the standard device of a nondisclosure clause in the contract of employment can be used. For software that is distributed, the software can be licensed rather than sold, where the licence requires the licensee not to reveal the "secret", i.e., not sell or give away copies of the software.

3. Patent

The strongest form of protection for software is patent protection. With a patent on an invention, the owner of the invention has two enormous advantages. First she has the right to the exclusive production, use and sale of the invention, or to license others to do these things. Second she has the right to exercise this exclusive control even if the invention is later reinvented by someone else. Clearly the possibility of obtaining a patent is an incentive to invent useful things that can be sold. But there is a deeper social purpose of patents. The way patent law works, in order to obtain a patent it is necessary for the invention to be sufficiently clearly described that anyone familiar with the technology involved can reproduce it. Moreover this description is made public in the process of patenting the invention. So not only are the individual's rights of possession established, but the existence of the invention and how it works are made public. In this way others are able to learn from and build on earlier inventions. But the original invention remains the exclusive property of the original inventor.

[The following paragraph was not in the lecture notes, it needs to be added. An extra mark or two can be given if the following points are made.]

The problem with patent law is that (at least so far) it does not apply to software on its own. The reasoning being that software is too much like a process of thought or a mathematical algorithm [1 mark]. *The only way in which software can be patented is if it is part of some kind of thing that can be patented, like some kind of machine which has a programmed or programmable operation* [1 mark].

(C)

Part (C) of the question should be answered in terms of consequentialist arguments. A good candidate will discuss why theories of natural rights do not to work. Both these discussions are in the following section of the distributed lecture notes:

[1 mark for saying why natural rights don't do the job - as follows:]

Natural rights

The argument for ownership:

The justification of ownership based on natural rights appeals to the idea that a person has a natural right to possess what she produces. The labour that she puts into her production is hers to begin with and thus the product of her labour should remain hers. The product would not have existed without her labour, it is composed, at least in part, of her labour. As such it should remain hers, she has a right to own it.

Problems with this argument:

In the case of software this argument seems to miss the point. If the product of a person's labour is a programme, then she still has it even if someone copies it. If I make a pot out of clay from the common, and you take it from me, then I don't have it any more. But if I write a programme and you copy it out of my files I still have it. So why do I object to your taking it in this way? The only objection can be in terms of some advantage I might gain by having exclusive use of it, in particular

the financial advantage I gain by selling it. But we can imagine a world where software never enters the commercial realm. We can easily imagine a world where software is published like scientific articles and the writer is rewarded in the same way as scientists are rewarded for their publications. In this world we would be highly motivated to publish our software. It seems that the morality of software ownership depends on the social system in which we live. In other words, software ownership, at least, seems not to be a natural right.

[2 marks for a defence of proprietary software on consequentialist grounds - as follows]
Consequences
The argument for ownership
The central motivation for maintaining the right to own software is the profit motive, and this is justified on the grounds that innovation will only come about if there is some advantage to be gained. On this basis, the right to own software is not a natural right but a social right that is justified in terms of its beneficial social consequences. The socially desirable consequences are progress and development of software. This progress and development, it is argued, will not take place unless those who make the effort to bring it about can see something in it for themselves. And that, it is claimed, is profit. The only way to guarantee profit for the producers of software is to give them control over the use and distribution of the software. They can then sell or license it for whatever profit they can get. It is argued that this system has the advantage that quality will be maximised through competition in the marketplace. Software of higher quality will naturally attract higher profits and so the sellers of software will strive to improve their software. This is the consequentialist argument for the social right to ownership.

[The last three parts of the question were addressed in both the lecture notes and the printed notes distributed to the students, though there was some variation between the two. In the following solution notes material from the printed notes is given first and then notes the lecturer used for the lectures are given in italics.]

(D) Arguments used in opposition to proprietary software (asterisks * indicate points for which a mark can be given - maximum of three marks):

In the early days of computing people developed software without the profit motive. The motive then was interest, curiosity, intellectual challenge, and, of course, need. Those who created useful software were rewarded within the computer community by the respect and appreciation of those who where able to benefit from the use of the software. In science this is still the prevailing way in which good work is rewarded. Publication of scientific work is the hallmark of success, the work is then available to the scientific community to use and build on as it sees fit. It is not hard to imagine a similar kind of reward system being used in the production of software.

The basic dilemma - incentive vs. progress:
In the late 1970s and early 1980s there was considerable concern about the extent of piracy and illegal software copying and the damage this might do to the software industry. The worry was that without the profit from software sales the industry would collapse and software development and progress would come to a halt. Now, in the 1990s the opposite kind of concern is being expressed - fears that too much ownership and control might interfere with software development. This illustrates the basic dilemma in thinking about software ownership. If too little ownership is allowed then development is unmotivated and if too much ownership is allowed then development is stifled.

If too little is owned then there is nothing for the potential developer to sell, there is no profit to motivate development. But if too much is owned then development is stifled for two different reasons. *The first reason is that building on previous developments requires licenses and agreements from those who own the software that embodies those previous developments, and this takes time and costs money. *The second reason is that to establish ownership of some new development a costly and time consuming search has to be made to establish that that development is not already owned. Either way, if very much of software is owned it costs time and money to make further developments.

[The following (in italics) are the lecturer's notes for the final lecture which did not get included in the printed notes that were distributed to the students:]

The Dilemma of Proprietary Software

*The ethical justification for the institution of proprietary software is the consequentialist justification that this is conducive to progress in development and improvement of software. *But it is also argued that quite the opposite consequences result - that the institution of proprietary software inhibits development and improvement of software. *It is argued that progress in development and improvement of software will take place by making software free and open to modification. For this latter condition to be fulfilled software must be supplied with its source programme - it must be open source software. This vision of open source free software is a revival of the original ethos of the computer community. It is most vociferously defended by Richard Stallman, founder of the Free Software Foundation. But Stallman is only the spokesperson for a significant movement that already exists within the computing community.*

*Richard Stallman's vision is idealistic, but not unrealistic. It is a vision of life with open-source, free software. In contrast to the self-interested competitive spirit of the proprietary software community, Stallman sees a community of co-operative sharing individuals where community spirit dominates the lust for money. Whereas the proprietary software community restricts wealth by restricting access to software, *the open-source, free software community increases wealth by making software widely available. *Stallman argues that software should not have owners, that there is no right of ownership for software because the owner does not lose anything if the software is copied. *In reply to the consequentialist argument that the owner loses income Stallman argues that most people who copy software would not have bought it anyway so there is really no economic loss. Stallman sees open-source, free software a step towards a future utopia in which there is no scarcity and everyone shares equally in the benefits of mass production.*

*Stallman sees many benefits deriving from the open-source, free software movement. *By developing software in an open and co-operative way progress is improved because there is no needless duplication of effort. *Also software support is easier to find - the user doesn't need to return to the producer, she can turn to any local software worker who can assess the problem by looking at the source code. If a genuine error is found in the source code that can be reported back to the community and all will benefit. *The movement will encourage people to learn about programming too - being able to see how things are done with computers will encourage people to develop their own programmes. *And last, but not least, the movement makes computing less expensive - the software is free and the service is more easily obtained.*

(E) Alternatives to proprietary software (asterisks * indicate points for which a mark can be given - maximum of 2 marks):

The Free Software Foundation offers an alternative to private ownership of software. The ideology of the FSF is that motivation should be interest, curiosity, intellectual challenge and need, and for this

reason software should not have owners. *To ensure that no one claims ownership of software that they produce, they attach a set of distribution terms referred to as copyleft (in contrast to copyright). In the FSF file, "Categories of Free and Non-Free Software", this is defined as follows:

*Copylefted software is free software whose distribution terms do not let redistributors add any additional restrictions when they redistribute or modify the software. This means that every copy of the software, even if it has been modified, must be free software.

*Indeed, a surprising amount of the internet depends on free software. *More than 63% of all websites are run by the Apache web server, free software developed by a team led by Brian Behlendorf. *Sendmail, the program that routes more than 75% of the internet's email is free software developed by Eric Allman and Greg Olson. *BIND, the software that provides the domain name service for the entire internet is free software developed by Paul Vixie. The flagship of the open-source, free software movement is, of course, * the Linux operating system. Linux has enjoyed a growing popularity in recent years and has even been adopted as an operating system by IBM.*

(F) How computer professionals can support themselves without proprietary software (asterisks * indicate points for which a mark can be given - maximum of two marks):
Of course if software developers were not to take their income from the marketplace, they would require another source of income. Funding for software developers would have to come from another source. In science this comes from government spending on universities and through research councils. *And it is not impossible to imagine a similar source of funding for software developers. But this would depend on further government spending and thus on the political climate. If the political climate favours private sector enterprise, then the science model of software development will not work.

[Further unprinted notes for the final lecture:]
*The main, and obvious, objection to the open-source, free software movement is that without the income generated from selling the software, the producers will not be able to carry on producing. In the GNU Manifesto, Stallman argues that there are ample sources of income for software workers. For example software workers can be employed to *install and port software, they can *teach computing skills, *offer help and advice services, and *provide maintenance services.*

*Stallman also suggests institutional structures that could be set up to support software development. For example, *money for software research could be raised by putting a tax on computers. This could then be distributed by a "Software Research Council" along the lines of the various science research councils. Allowances toward this tax could be made for those who make software donations to the community.*

[The following discussion was included in the lecture notes but was not covered in the lectures. It contains nothing of relevance to the exam question that has not already been covered:]

The morality of copying software
It seems then that the morality of copying software is not a matter of some kind of ultimate objective right or wrong, but is a matter of consequences within a particular social system. For that reason we cannot simply decide through reason alone whether it is morally right to freely copy software but, rather, we must consider what this means in our own social context. The fundamental principle of this discussion so far has been that progress in the development of software is a good and desirable end. Our assessment of the consequences of software ownership has been based on this principle.

But there is an intrinsic dilemma that seems to arise from this principle. That is that ownership of software both serves as an incentive for progress and serves to inhibit progress.

Is it wrong to copy proprietary software?
Given that there are laws against copying proprietary software, the burden of argument rests with those who claim that it is morally justified to do so. The major argument in favour of copying proprietary software is that the laws forbidding it are bad laws, they are unnatural, unjust or they inhibit progress in software development.

There are two kinds of situation in which it is morally appropriate to break the law, the general situation and the particular situation. The general situation where breaking the law is morally appropriate arises when the general consequences of obeying the law are positively harmful, or unjust. In these cases we can justify breaking the law on the grounds that we have an overriding moral obligation to overcome the injustice, or to reduce the harm, that would result from obeying the law. Particular situations where breaking the law is morally appropriate arise when immediate moral demands override the general moral demand that we obey the law.

Now we can ask whether breaking the law to copy software corresponds to either of these two situations. It may be that there are particular circumstances in which there are overriding moral reasons for copying proprietary software, but each of these must be considered on its own merits. Here there is a matter of judgement. Certainly we can imagine situations where serious harm could be prevented by illegally copying some software, and we can also see situations where very little or no loss at all would result from not doing so. The judgement has to be made as to where the situation changes - we have to be able to tell just how serious the harm needs to be before we are justified in making illegal software copies to prevent it.

However it can also be argued that copying software is justified in general. It can be argued that the laws making software a kind of property are wrong - they are unjust or have harmful consequences. The sorts of harm that could be done have been considered already - software development might be inhibited by being bound up under licensing agreements, or a useful programme that someone creates may turn out to have already be written and owned by someone else. But the argument for breaking laws in general depends on the laws having positively harmful consequences, or being positively unjust. If the problem is simply that we can see how things might be better but do not think that things are positively bad, then the argument doesn't work.