PAPER-I

Why post this?

Over the past couple of years, I've interviewed a lot of folks for development positions - mostly C++ and COM on Windows. Some people are outstanding, some are truly horrible; but a large number of people seem bright, but just don't have the depth of knowledge that I think a professional C++ developer should have.
What follows is a list of questions that I typically ask interview candidates. No, I'm not going to give you the answers - that's up to you. Someday, you'll be sitting across from me or someone like me, and they'll ask you questions like these.
When that happens, resist the temptation to give a pat, memorized answer. You should be able to take any one of these questions, and talk about each one with an interviewer for at least five minutes or so. The type of candidate that really impresses me is not the one who can answer using the fewest words, it's the one that knows something well, and can explain it to me.
IT's very likely that some of these quesitons might seem unclear, for whatever reason - most likely, my inability to frame them properly in such a constrained format. If you have any comments, or suggestions on how to improve this list, please feel free to contact me at srobb@icubed.com.

General Questions

These are questions I tend to ask at the beginning or at then end of an interview. There are no right or wrong answers - the intent is to find out more about a candidate, and to get them comfortable talking about themselves.

- How did you get into computer science?
- What kind of technical publications (print or online) do you read on a regular basis?
- What was the last book you read (does not have to be job related!)
- If you could recommend one resource (book, web site, etc.) to a new software developer just out of school, what would it be?
- What was the most interesting project that you worked on?
- What was the most challenging project that you worked on?
- If you could have any job in the world, what would it be?

C++ Questions

- What are the major differences between C and C++?
    - What are the differences between new and malloc?
    - What is the difference between delete and delete[]?
    - What are the differences between a struct in C and in C++?
    - What are the advantages/disadvantages of using #define?
    - What are the advantages/disadvantages of using inline and const?
- What is the difference between a pointer and a reference?

    - When would you use a pointer? A reference?
    - What does it mean to take the address of a reference?
- What does it mean to declare a function or variable as static?
- What is the order of initalization for data?
- What is name mangling/name decoration?
    - What kind of problems does name mangling cause?
    - How do you work around them?
- What is a class?
    - What are the differences between a struct and a class in C++?
    - What is the difference between public, private, and protected access?
    - For class CFoo { }; what default methods will the compiler generate for you>?
    - How can you force the compiler to not generate them?
    - What is the purpose of a constructor? Destructor?
    - What is a constructor initializer list?
    - When *must* you use a constructor initializer list?
    - What is a:
        - Constructor?
        - Destructor?
        - Default constructor?
        - Copy constructor?
        - Conversion constructor?
    - What does it mean to declare a...
        - member function asvirtual?
        - member function asstatic?
        - member function asstatic?
        - member varible as static?
        - destructor as static?
    - Can you explain the term "resource acqusition is initialization?"
    - What is a "pure virtual" member function?
    - What is the difference between public, private, and protected inheritance?
    - What is virtual inheritance?
    - What is placement new?
    - What is the difference between operator new and the new operator?
- What is exception handling?
    - Explain what happens when an exception is thrown in C++.
    - What happens if an exception is not caught?
    - What happens if an exception is throws from an object' s constructor?
    - What happens if an exception is throws from an object' s destructor?
    - What are the costs and benefits of using exceptions?
    - When would you choose to return an error code rather than throw an exception?
- What is a template?
- What is partial specialization or template specialization?
- How can you force instantiation of a template?
- What is an iterator?

- What is an algorithm (in terms of the STL/C++ standard library)?
- What is std::auto_ptr?
- What is wrong with this statement? std::auto_ptr ptr(new char[10]);
- It is possible to build a C++ compiler on top of a C compiler. How would you do this? (Note: I' ve only asked this questin once; and yes, he understood that I was asking him how cfront was put together. We hired him.)/i>

Code Snippets

I like to put these up on a whiteboard, and ask questiosn about them.

What output does the following code generate? Why?
What output does it generate if you make A::Foo() a pure virtual function?

```
#include <iostream>

class A {
public:
        A() {
 this->Foo();
}
        virtual void Foo() {
 cout << "A::Foo()" << endl;
}
};

class B : public A {
public:
        B() {
 this->Foo();
}
        virtual void Foo() {
 cout << "A::Foo()" << endl;
}
};

int main(int, char**)
{
        B       objectB;

        return 0;
}
```

What output does this program generate as shown? Why?

```
#include <iostream>
```

```
class A {
        public:
                A() {
         cout << "A::A()" << endl;
    }
                ~A() {
         cout << "A::~A()" << endl; throw "A::exception";
    }
        };

        class B {
        public:
                B() {
         cout << "B::B()" << endl; throw "B::exception";
    }
                ~B() {
         cout << "B::~B()";
    }
        };

        int main(int, char**) {
                try {
                        cout << "Entering try...catch block" << endl;

                        A       objectA;
                        B       objectB;

                        cout << "Exiting try...catch block" << endl;
                } catch (char* ex) {
                        cout << ex << endl;
                }

                return 0;
        }
```

Misc. Questions

- What' s the difference between SQL, DDL, and DML?
- What' s a join? An inner join? An outer join?
- Describe HTTP.
- What's a design pattern?
- Can you explain the singleton, vistor, facade, or handle class design pattern?

PAPER-II

C/C++ Interview Questions

The following article is taken from the Dr. Dobb's Journal, September 1996. It recommends suitable methods for interviewing C++ job applicants.

Wanted: Senior C++ Programmer

by Al Stevens

Good morning, Mr. Phelps. The title of this month's column is a typical entry taken from the help-wanted section of the local classified ads. Does it get your attention? Add your address and phone number to the ad, publish it in your local paper, and, depending on where you live, you will get lots of calls and mail. Headhunters will deluge you with résumés, every one of which has the token C++ somewhere near the top of the first page. Programmers who want to reap the harvest of the C++ demand will polish up their résumés to emphasize their C++ exposure and send them in.

Your mission, should you choose to accept it, is to find the best applicant, the one capable of filling the senior C++ programming position, from among the many.

Last month, I listed some questions that you can ask people who apply for a C++ programming job. This month I'll discuss what I think are good answers to those questions. I'm writing this column in June, and the August issue is not on the street yet, so, although I asked for your comments, you haven't had that opportunity. Once again, this technique is my own device, is influenced by my opinions, and needs polish. I'm eager to hear all comments and criticisms.

The questions are in three categories:

* The first category probes whether applicants have a rudimentary understanding of the differences between C and C++.
* The second category examines their grasp of class design.
* The third determines how well they have kept abreast of language changes that the ANSI committee has proposed and approved.

I developed this technique after watching a client conduct interviews of many applicants for a single position. It became obvious that there was no structure to the art of the interview. The interviewer understood the problems of the project, but she is not a C++ programmer. Consequently, her technical questions were superficial ("Have you ever programmed with OWL?", for example) and the answers revealed nothing important about the applicant's abilities. When she asked me to help, I realized after a couple of interviews that my extemporaneous style needed better organization. I needed a plan. This set of questions is the result.

The first question you should ask is if the applicant is a devoted reader of this column. If so, you're on your own. The programmer has probably read and memorized these answers by now. Hire 'em. Readers of this column are always a good bet, I bet.

After that, the questions you ask are determined by how applicants represent their knowledge and experience. Anyone who really wants the job tries to match aspects of their experience with an interviewer's questions. It's a natural tendency.
There is an art to résumé writing, too, that emphasizes work experiences to fit the job requirements no matter how trivial the experience was. Some headhunters are really good at it. Therefore, don't depend on the details of a résumé to properly identify an applicant's abilities. But you do need to make that determination because if you ask too many questions that are beyond the scope of a programmer's knowledge, the interview deteriorates into a confrontation, and the applicant stops wanting to work for you. Consequently, the first three questions are designed to reveal how applicants want their skills to be regarded and should permit you to follow with the appropriate questions chosen from the three groups.

Questions to Qualify an Applicant

Q: Do you have a basic understanding of C and C++ and their similarities and differences?
A: Yes.
If the answer is "yes," proceed and plan to ask all the questions in the first group. If not, the interview for a C++ programmer's job should be over, and you should be talking about other employment opportunities.
Q: Have you participated in the design of C++ classes to support an application's problem domain?
A: Yes or No.
If the answer is "yes," plan to include the questions in the second group. Applicants usually want to describe the details of the systems they have designed. Pay attention if you understand the nature of the application. If not, pretend to pay attention. If they have not actually done any class design, ask if they think they understand it. If so, include the second set of questions.
Q: Do you keep up with what the ANSI C++ Standards committee is doing?
A: Yes or No.
Most people do not. Few people have the time. But occasionally there is that rare soul who reads all the magazines and books, owns a copy of the draft standard, and regularly tracks the C++ news groups on the net. If an applicant claims to be one of them, include the third group of questions in your interrogation.

The Art of the Interview

With your applicants qualified as to how they represent themselves, ask the questions from the chosen three groups. Mix the questions and keep a light but detached attitude going if you can. As soon as you get into details, many people tighten up. They know they are being tested, and as they try to interpret the question to figure out what you want to hear, sometimes they lose sight of the objective, which is to expose what they know rather than what their intuition tells them is an acceptable answer.
Maintain your detachment. Be cordial, but don't get chummy just yet. Remember, if you hire this person to work with you on your typically understaffed, under- budget, overdue

C++ programming project, the two of you are going to spend a lot of tense moments together. Observe how the applicant handles the pressure of the interview. If your shop is typical, the two of you will have to routinely deal with a lot more pressure than this. Don't expect everyone to get everything right (unless they read this column, of course, and fibbed about the first question).

Questions for All Applicants

Here's the first group of questions and my opinions about what some acceptable answers would be. These questions do not cover C++ wall-to-wall, of course. I selected them as typical of the kind of knowledge that all C++ programmers should be expected to possess. There are five questions. Three correct answers is a good score.

Q: How do you link a C++ program to C functions?

A: By using the extern "C" linkage specification around the C function declarations.

Programmers should know about mangled function names and type-safe linkages. Then they should explain how the *extern "C"* linkage specification statement turns that feature off during compilation so that the linker properly links function calls to C functions. Another acceptable answer is "I don't know. We never had to do that." Merely describing what a linker does indicates that the programmer does not understand the issue that underlies the question.

Q: Explain the scope resolution operator.

A: It permits a program to reference an identifier in the global scope that has been hidden by another identifier with the same name in the local scope.

The answer can get complicated. However, it should start with "::". If the programmer is well into the design or use of classes that employ inheritance you might hear a lot about overriding member function overrides to explicitly call a function higher in the hierarchy. That's good to know, but ask specifically about global scope resolution. You're looking for a description of C++'s ability to override the particular C behavior where identifiers in the global scope are always hidden by like identifiers in a local scope.

Q: What are the differences between a C++ *struct* and C++ *class*?

A: The default member and base-class access specifiers are different.

This is one of the commonly misunderstood aspects of C++. Believe it or not, many programmers think that a C++ *struct* is just like a C *struct*, while a C++ class has inheritance, access specifiers, member functions, overloaded operators, and so on. Some of them have even written books about C++. Actually, the C++ *struct* has all the features of the *class*. The only differences are that a *struct* defaults to public member access and public base-class inheritance, and a *class* defaults to the private access specifier and private base-class inheritance. Getting this question wrong does not necessarily disqualify an applicant. Getting it right is a definite plus.

Saying, "I don't know" is definitely the wrong answer. I advance an unusual position about this. C++ programmers should at least believe that they know the differences, even when they are wrong about them. Getting it wrong is, therefore, right. You can explain the true difference in the interview and advance the programmer's knowledge. If they disagree vociferously, you have an opportunity to observe how they handle contentious debate when they are wrong and don't know it yet.

Q: How many ways are there to initialize an *int* with a constant?

A: Two.
There are two formats for initializers in C++ as shown in the example that follows. The first format uses the traditional C notation. The second format uses constructor notation.
int foo = 123;
int bar (123);
It's acceptable when a programmer does not know about the second notation, although they should certainly know about the first one. Many old-timer C programmers who made the switch to C++ never use the second idiom, although some wise heads of C++ profess to prefer it. If your applicant is quick with the right answer, that's a good sign.
Q: How does throwing and catching exceptions differ from using *setjmp* and *longjmp*?
A: The throw operation calls the destructors for automatic objects instantiated since entry to the *try* block.
Exceptions are in the mainstream of C++ now, so most programmers, if they are familiar with *setjmp* and *longjmp*, should know the difference. Both idioms return a program from the nested depths of multiple function calls to a defined position higher in the program. The program stack is "unwound" so that the state of the program, with respect to function calls and pushed arguments, is restored as if the calls had not been made. C++ exception handling adds to that behavior the orderly calls to the destructors of automatic objects that were instantiated as the program proceeded from within the *try* block toward where the *throw* expression is evaluated.
Applicants might think you want to hear about the notational differences between the two idioms. Let them proceed to explain the syntax of *try* blocks, *catch* exception handlers, and *throw* expressions. Then ask them specifically what happens in a *throw* that does not happen in a *longjmp*. Their answer should reflect an understanding of the behavior described in the previous answer.
One valid reason for not knowing about exception handling is that the applicant's experience is exclusively with older C++ compilers that do not implement exception handling. I would prefer that they have at least heard of exception handling, though. Another marginally acceptable reason is that their former supervisors and designers did not mandate and specify the use of exception handling in programs. In that case get the names of those supervisors and designers so that you can decline their applications if they should come a' knocking.
It is not unusual for C and C++ programmers to be unfamiliar with *setjmp*/*longjmp*. Those constructs are not particularly intuitive. A C programmer who has written recursive descent parsing algorithms will certainly be familiar with *setjmp*/*longjmp*. Others might not, and that's acceptable. In that case, they won't be able to discuss how *setjmp*/*longjmp* differs from C++ exception handling, but let the interview turn into a discussion of C++ exception handling in general. That conversation will reveal a lot about a programmer's understanding of C++.

Questions for Class Designers

The second group of questions explores the applicant's knowledge of class design. There are eight questions. Five out of eight is a good score.
Q: What is your reaction to this line of code?

delete this;

A: It's not a good practice.

Many applicants will look at you like you are nuts. They've never heard of this usage, and it's never occurred to them. That's a very good answer. Perhaps they will try to explain the behavior of the statement. Ask them to contemplate its consequences. Two quite acceptable reactions are, "Don't do it," and "Don't do it unless you really know what you are doing and you are a masochist."

A good programmer will insist that you should absolutely never use the statement if the class is to be used by other programmers and instantiated as static, extern, or automatic objects. That much should be obvious.

The code has two built-in pitfalls. First, if it executes in a member function for an extern, static, or automatic object, the program will probably crash as soon as the *delete* statement executes. There is no portable way for an object to tell that it was instantiated on the heap, so the class cannot assert that its object is properly instantiated. Second, when an object commits suicide this way, the using program might not know about its demise. As far as the instantiating program is concerned, the object remains in scope and continues to exist even though the object did itself in. Subsequent dereferencing of the pointer can and usually does lead to disaster. I think that the language rules should disallow the idiom, but that's another matter.

In *More Effective C++* (Addison-Wesley, 1996), Scott Meyers devotes one of his items to "*delete this*," implying that there are valid applications for the idiom and advancing contrived code kludges to make it seem to work better. A programmer who has read this otherwise very good book might think that the practice is acceptable. Experience leads me to disagree.

Q: What is a default constructor?

A: A constructor that has no arguments.

If you don't code one, the compiler provides one if there are no other constructors. If you are going to instantiate an array of objects of the class, the class must have a default constructor.

Q: What is a conversion constructor?

A: A constructor that accepts one argument of a different type.

The compiler uses this idiom as one way to infer conversion rules for your class. A constructor with more than one argument and with default argument values can be interpreted by the compiler as a conversion constructor when the compiler is looking for an object of your constructor's type and sees an object of the type of the constructor's first argument.

Q: What is the difference between a copy constructor and an overloaded assignment operator?

A: A copy constructor constructs a new object by using the content of the argument object. An overloaded assignment operator assigns the contents of an existing object to another existing object of the same class.

First, the applicant must know that a copy constructor is one that has only one argument of the same type as the constructor. The compiler invokes a copy constructor wherever it needs to make a copy of the object, for example to pass an argument by value. If you do not provide a copy constructor, the compiler creates a member- by-member copy constructor for you.

You can write overloaded assignment operators that take arguments of other classes, but that behavior is usually implemented with implicit conversion constructors. If you do not provide an overloaded assignment operator for the class, the compiler creates a default member- by-member assignment operator.

This discussion is a good place to get into why classes need copy constructors and overloaded assignment operators. If the applicant discusses these with respect to data member pointers that point to dynamically allocated resources, the applicant probably has a good grasp of the problem.

Q: When should you use multiple inheritance?

A: There are three acceptable answers: "Never," "Rarely," and "When the problem domain cannot be accurately modeled any other way."

There are some famous C++ pundits and luminaries who disagree with that third answer, but I will accept it.

Let's digress to consider this issue lest your interview turn into a religious debate. Consider an *Asset* class, *Building* class, *Vehicle* class, and *CompanyCar* class. All company cars are vehicles. Some company cars are assets because the organizations own them. Others might be leased. Not all assets are vehicles. Money accounts are assets. Real estate holdings are assets. Some real estate holdings are buildings. Not all buildings are assets. Ad infinitum. When you diagram these relationships, it becomes apparent that multiple inheritance is a likely and intuitive way to model this common problem domain. The applicant should understand, however, that multiple inheritance, like a chainsaw, is a useful tool that has its perils, needs respect, and is best avoided except when nothing else will do.

Q: What is a virtual destructor?

A: The simple answer is that a virtual destructor is one that is declared with the virtual attribute.

The behavior of a virtual destructor is what is important. If you destroy an object through a pointer or reference to a base class, and the base-class destructor is not virtual, the derived-class destructors are not executed, and the destruction might not be complete.

Q: Explain the ISA and HASA class relationships. How would you implement each in a class design?

A: A specialized class "is" a specialization of another class and, therefore, has the ISA relationship with the other class. An *Employee ISA Person*. This relationship is best implemented with inheritance. Employee is derived from *Person*. A class may have an instance of another class. For example, an employee "has" a salary, therefore the *Employee* class has the HASA relationship with the *Salary* class. This relationship is best implemented by embedding an object of the *Salary* class in the *Employee* class.

The answer to this question reveals whether the applicant has an understanding of the fundamentals of object- oriented design, which is important to reliable class design.

There are other relationships. The USESA relationship is when one class uses the services of another. The *Employee* class uses an object (*cout*) of the *ostream* class to display the employee's name on the screen, for example. But if the applicant gets ISA and HASA right, you don't need to go any further.

Q: When is a template a better solution than a base class?

A: When you are designing a generic class to contain or otherwise manage objects of other types, when the format and behavior of those other types are unimportant to their

containment or management, and particularly when those other types are unknown (thus, the genericity) to the designer of the container or manager class.
Prior to templates, you had to use inheritance; your design might include a generic *List* container class and an application-specific *Employee* class. To put employees in a list, a *ListedEmployee* class is multiply derived (contrived) from the *Employee* and *List* classes. These solutions were unwieldy and error-prone. Templates solved that problem.

Questions for ANSI-Knowledgeable Applicants

There are six questions for those who profess knowledge of the progress of the ANSI committee. If you claim to have that much interest in the language, you should know the answers to all these questions.
Q: What is a mutable member?
A: One that can be modified by the class even when the object of the class or the member function doing the modification is *const*.
Understanding this requirement implies an understanding of C++ *const*, which many programmers do not have. I have seen large class designs that do not employ the *const* qualifier anywhere. Some of those designs are my own early C++ efforts. One author suggests that some programmers find *const* to be such a bother that it is easier to ignore *const* than to try to use it meaningfully. No wonder many programmers don't understand the power and implications of *const*. Someone who claims to have enough interest in the language and its evolution to keep pace with the ANSI deliberations should not be ignorant of *const*, however.
Q: What is an *explicit* constructor?
A: A conversion constructor declared with the explicit keyword. The compiler does not use an *explicit* constructor to implement an implied conversion of types. It's purpose is reserved explicitly for construction.
Q: What is the Standard Template Library?
A: A library of container templates approved by the ANSI committee for inclusion in the standard C++ specification.
A programmer who then launches into a discussion of the generic programming model, iterators, allocators, algorithms, and such, has a higher than average understanding of the new technology that STL brings to C++ programming.
Q: Describe run-time type identification.
A: The ability to determine at run time the type of an object by using the *typeid* operator or the *dynamic_cast* operator.
Q: What problem does the namespace feature solve?
A: Multiple providers of libraries might use common global identifiers causing a name collision when an application tries to link with two or more such libraries. The namespace feature surrounds a library's external declarations with a unique namespace that eliminates the potential for those collisions.
This solution assumes that two library vendors don't use the same namespace identifier, of course.
Q: Are there any new intrinsic (built-in) data types?
A: Yes. The ANSI committee added the *bool* intrinsic type and its *true* and *false* value keywords.

Other apparent new types (string, complex, and so on) are implemented as classes in the Standard C++ Library rather than as intrinsic types.

What Not to Do

Don't ask the applicant to write a program on the spot. Many programmers, myself included, like to ponder a program for a while before writing any code. We would consider a one-hour deadline as a threat. We would be unwilling to let anyone see such a program. You'll get a far better understanding of the applicant's programming skills by asking a former employer about the programmer's ability to write reliable, extensible, maintainable, and timely code that fulfills the user's requirements.

Don't turn these questions into a written test. Some people don't take tests well, particularly when the answers involve narrative text. Many fine programmers are not articulate writers. Others simply hate tests and shut down mentally.

Case in point. My pal Bill Chaney was the highest-time student pilot I ever met. Over a period of several years, he accumulated close to 1000 hours on his student pilot's license. I would have trusted Bill to fly one of my children anywhere in that old Cessna 150 of his, but he was intimidated by the environment of the written examination and could not pass it. There's a happy ending: Several of us ganged up on Bill, crammed him on the written exam, saturated him with a positive mental attitude, and railroaded him through passage of the test. Bill is a legal pilot today.

There's another reason to avoid the written test, one that is driven by social changes in our culture. Last month I said, "For some reason, the Labor Department doesn't want you disqualifying anyone based on their inability to demonstrate that they can do the job." Now I'll expand on that statement, and perhaps stir a bit of controversy.

During my government client's quest for employees, she was questioned by the local EEO representative about her testing practices. They were particularly concerned about whether she was administering, and specifically, grading tests. There is a strong belief that certain segments of our society, like my pal Bill Chaney, do not take tests well, that those segments can be identified by ethnic divisions, and that testing them provides the potential for discrimination. That belief, or the fear of those who hold it, drives official policy.

The EEO is not consistent in this concern. You can't carry passengers in a 747 without proving your experience and ability and passing several of the government's oral and written graded tests. You don't get anywhere near my gall bladder, retirement account, or household wiring without a certificate on the wall proving that you have scored passing grades on a bevy of written tests. The potential to discriminate is overlooked when something important like our lives or our money is at stake.

My parents' generation was the last one that discriminated against ethnic and gender boundaries with the sanction of law to back them up. (My parents didn't do it, but their generation did and the law allowed it.) Many of my generation inherited and retained that bias even when the sanctions were removed through legislation and judicial decree. Most of the newer generation is free of that bias, I hope, at least intellectually if not emotionally. But the swing of the pendulum is not yet dead center. If you administer written tests to programmers, grade those tests, and use those grades to disqualify

someone for employment, you might find yourself the object of a discrimination investigation.

So, ask those questions and take notes. But don't write grades on a score sheet. Or, in case you do, there is a device in our office named "Hillary." Get one. (It's also called a shredder.) As usual, if you or any of your interviewers are killed or captured, *DDJ* will disavow any knowledge of your activities.

PAPER - III

Few people know as much about smart cards as Patrice Peyret, founding president and CEO of Integrity Arts, a start-up company focused on smart cards, which was acquired by Sun Microsystems in September.

Peyret, a renowned smart card expert, has been working with smart cards since 1987, when he worked for Thompson Consumer Electronics on the world' s first set-top box equipped with a smart card. More than 3 million of these BSkyB set-top boxes are in use in the UK. More recently (1989 to 1995), Peyret was head of research and development for Gemplus, the world' s largest smart card company.

As a result of Sun' s acquisition of Integrity Arts, Peyret is now Director of JavaSoft' s Consumer Transactions group, which focuses on both JavaCards and Java Commerce Clients -- electronic commerce software running on PCs, network computers (NCs), and possibly set-top boxes.

In late October, *JavaWorld* columnist Rinaldo Di Giorgio talked at length with Peyret about the state of smart cards, the Java cards, and his latest projects. Read on to learn Peyret' s answers to the following questions:

1. What are smart cards?
2. How do they work?
3. Are there any significant applications of smart cards?
4. Can you explain why smart cards are not common in the U.S.?
5. So smart cards have been around for quite a long time?
6. What is JavaCard?
7. So JavaCard has had an effect on the smart card industry?
8. So developing applications for smart cards was difficult?
9. How is JavaCard going to improve this?
10. Is there any danger that JavaCard is a passing fad?
11. How do you identify an application that would benefit from using a smart card?
12. Why are smart cards always associated with security?
13. You mentioned the carrying of personal data...
14. Is there room for small developers in the smart card world?
15. Does JavaCard have any competition?
16. Should Microsoft consider licensing JavaCard?
17. What is JavaSoft doing to ensure access to smart cards on many platforms.
18. How will multiple applications co-exist on JavaCard?
19. What is the difference between OpenCard and PC/SC?
20. What are some of these guidelines?
21. What is your group working on at JavaSoft?
22. What are some of the tools and when will they be available?
23. Why are you so sure JavaCard is going to be the largest deployed computing platform in the world?

JavaWorld: What are smart cards?
Peyret: A smart card is typically a "credit card" sized form factor with a small-embedded computer chip. This card-computer can be programmed to perform tasks and store information. There are different types of smart cards: memory cards, processor cards, electronic purse cards, security cards, and JavaCards.
JW: How do they work?
Peyret: Very simply, a smart card that has a processor is inserted into a smart card reader (commonly called a card terminal) and is available for use. The software wishing to communicate with the reader needs to send some commands to manage the reader, things like power up and transfer command to card. The commands sent to cards can be custom, but we prefer to use the standard ISO 7816 specifications, which define command formats in great detail. Many different types of readers exist and soon we hope to see them shipped as standard equipment on PCs. They are not as uncommon as you think: There are several million readers deployed in homes in the United States in the Digital Satellite Systems (DSS) units.
JW: Are there any significant applications of smart cards?
Peyret: There are many significant smart card applications.

- Banks: Small trials in the U.S.; entire countries using the card in Europe and places like South Africa.
- Medical applications: In Germany 80 million people can use smart cards when they go to the doctor.
- Voting: In Sweden you can vote with your smart card, which serves as a non-repudiation device.
- Entertainment: Most DSS dishes in the U.S. have smart cards.
- Telecommunications: Many cellular phones come with smart cards in Europe and will soon be shipping in the United States.
- Mass Transit: British Air relies on rail and air connections more than most airports. There were many delays because customers could not be tracked while they were in transit, so no one knew where the customers were, which caused aircraft to be held for phantom customers. To solve this problem, British Air gives passengers contactless smart cards, and radio receivers track them throughout the facility. Now flights only wait when necessary, controllers can be given estimated ready times, and new departure slots can be calculated.

JW: Can you explain why smart cards are not common in the U.S.?
Peyret: Smart cards have been very popular in Europe, Asia. There have been huge smart card deployments in Japan, South Africa and Germany. Recently in the U.S. these cards have started to appear in pilots as a debit card. There are currently 650 million smart cards deployed in the world. In a few years I anticipate that there will be one billion cards deployed in the world. smart card technology matured outside of the U.S. due to U.S. infrastructure and cultural reasons. The U.S. infrastructure is very different than Europe, or Japan or China for example. In the U.S. there are 14,000 banks and several hundred telecommunication service companies, and U.S. users culturally have preferred a pay-later approach, which required a good communication infrastructure. In other countries, merchants are not always online so the smart card also solved a communication

infrastructure problem. In the U.S. it is harder to get the banks to cooperate on one standard. The same goes for the telecommunication companies. Many countries in Eastern Europe are going directly to chip cards; this approach also is taken by China.
JW: So smart cards have been around for quite a long time:
Peyret: Smart cards are a relatively mature solution in the process of manufacturing cards, but relatively new to the field of integrating into information systems. JavaCard provides a win/win situation by reducing the time to market and offering modern APIs that can be used to interface to corporate and consumer systems while at the same time Java gets a platform that has billions of deployed units.
JW: What is JavaCard?
Peyret: JavaCard is a smart card that is capable of running Java byte codes. Consider what this means as the cards become more powerful, that little card will be able to run some of the applications you run on your personal computer. We have no reason to believe that this will not be the case. We may face limits on the credit card form factor since this means more powerful chips can't be used due to the packaging constraints such as flexibility, but JavaCard should not be viewed as a credit-card-only standard. As new devices emerge, JavaCard should be considered as a design option.
JW: So JavaCard has had an effect on the smart card industry?
Peyret: So far Java has been relatively isolated without much relation to previous standards, bridging between the two worlds: The card world and the Java world. Prior to Java the smart card industry suffered from two bottlenecks:

1. Small group of knowledgeable people to program the cards, which exacerbated the time-to-market problem.
2. Lack of ubiquity at the application level.

JW: So developing applications for smart cards was difficult.
Peyret: Yes. It was relegated to a small group of individuals practicing an arcane art. Development tools that you could buy in a store did not exist. With JavaCard I expect that you will be able to buy development tools in a store. We are developing a JavaCard applet/application developers guide so that applications can be developed for a JavaCard.
JW: How is JavaCard going to improve this?
Peyret: Programming smart cards applications is very different than programming regular applications on a PC. The two most difficult aspects of this man-machine interfaces are the lack of large memory spaces typically found on workstations and PCs and the difficulty of debugging on the smart card, which can just go quiet for bugs quite often.
Smart card applications need to be designed to account for a small amount of memory and limited processing power of chips.
JW: Is there any danger that this is a passing fad?
Peyret: Unlikely. Perhaps the form factor may change, but as time goes on I think you will carry many computers at all times, perhaps in your watch or pen as well as your wallet. I really view JavaCard as a concept ... that I think will become more popular as smart cards are understood: The concept of wearable computers. We see this already in things like the ["iButton" smart ring] from Dallas Semiconductor, which will eventually

support Java and can be worn as a piece of jewelry, or the contactless smart cards that you can stick in your wallet.
JW: How do you identify an application that would benefit from using a smart card?
Peyret: Typically [...] any application requiring authentication or some portable memory can benefit from a smart card. Smart cards can be used for authentication and as a secure, convenient, portable storage mechanism. Developers no longer are bound by the lack of development environments.
JW: Why are smart cards always associated with security?
Peyret: One of the fundamental problems in securing computer systems is the need for tamper-resistant storage of keys. Smart cards provide this functionality as well as the ability to upgrade and/or replace a security solution when it becomes compromised. For example, there are millions of digital satellite systems that are smart card-enabled, and if some enterprising hacker cracks the security, the millions of DSS units need not be replaced; we can just mail out new cards. With JavaCard it gets even better in that we just send new cardlets [JavaCard applications] to everyone.
JW: You mentioned the carrying of personal data...
Peyret: Smart cards are really excellent at carrying your personal data. Think of it as a digital wallet that can store all this personal date under electronic lock and key.
JW: Is there room for small developers in the smart card world?
Peyret: Consider the need for applications as smart cards begin to participate in each of our daily lives as we interact with our latest PDA, phone, set-top box, NC, smart card enabled ATM and, of course, buy coffee at Starbucks. I can see a market where there are thousands of small cardlets for all the different functions you need, for example. Consider the number of cardlets required for smart card-enabled doors, cars, and entire buildings.
JW: Does JavaCard have any competition?
Peyret: As of this time the companies responsible for producing 80 percent of the cards have licensed JavaCard.
JW: Should Microsoft consider licensing JavaCard?
Peyret: Not really JavaCard is usually licensed by card Manufacturers. Microsoft does not currently produce cards.
JW: What is JavaSoft doing to ensure access to smart Cards on many platforms.
Peyret: My group is working with the licensees to make sure all the licensees are compatible on the 2.0 specification. We do this by providing support and providing test cases as well as reference implementations.
JW: How will multiple applications co-exist on JavaCard?
Peyret: Often security solutions are very good but very useless. [Systems become] so secure that no one uses them. We are taking a more practical approach with JavaCard and using the GateWay Security Model described to allow multiple applications to exchange data. It would be pretty foolish to present several smart cards to perform a transaction, although in all fairness we already provide several credit cards to perform many common transactions and maybe this model will stay, car companies and airline companies will issue different cards.
JW: What is the difference between OpenCard and PC/SC?
Peyret: OpenCard is a collection of companies endorsing a Java interface to smart cards. OpenCard is written in Java and has the potential to be 100% pure. JavaSoft will soon be releasing an API that supports Serial and Parallel devices. Once this is available,

OpenCard can be 100% pure. But we are diverging. OpenCard is a fairly lightweight interface to smart cards that provides a framework for developing smart card applications in Java. PC/SC is an interface to smart cards for Windows platforms. OpenCard, in the spirit of cooperation, has provided an interface layer so that OpenCard can use PC/SC to basically manage smart card readers or terminals and send APDUs [application protocol data units] to these readers. So if you write your application to the OpenCard specification you will get portability if you write it to PC/SC you will only work on Windows platforms, which may not be the predominant personal platform.
JW: What are some of these guidelines?
Peyret: Developers should be able to buy the tools in a store, buy a few cards in the store, and read the latest column in *JavaWorld* and then build an applet for the Card. In the past this was not possible, programs were burned into the chip and the special chips had to be produced. Smart cards can become a $16 billion market.
JW: What is your group working on at JavaSoft?
Peyret: My group is concentrating on getting the 2.0 specification completed and accepted as well as developing reference implementations and tools and of course providing support to licensees.
JW: What are some of the tools and when will they be available?
We are working on some specific tools to assist developers in developing smart card applications. A smart card has a pretty restrictive interface with a difficult user machine interface. We are working on:

- card simulators,
- debuggers, [and]
- terminal simulators

for host systems. We do not expect these tools to be available until Q1/98. Debugging smart card applications can be more difficult than debugging, say, an embedded controller, because with a controlled you could always get an in-circuit emulator or ICE. Due to the security constraints of a smart card, you can' t replace the CPU easily.
JW: Why are you so sure JavaCard is going to be the largest deployed computing platform in the world?
Peyret: There is this interesting phenomena, call it exponential growth or powers of 10 or even geometric growth. It goes something like this: In the beginning there were one or two electronic computers then there were 10 times as many IBM 7xxx followed by 10 times as many mainframes followed by 10 times as many minicomputers followed by 10 times as many PCs followed by 10 times as many Java Cards. Given these many platforms there is a need for developers. When cards reach a billion units, that is a rather large audience with quite a bit of ubiquity.

PAPER –IV

Some typical interview questions

Every interview is different, but often the same types of questions come up...

There are some questions that regularly come up in interviews. By going through the list below and **PREPARING** real answers to each of the questions (you can even write them out to make sure) you will be ready to face some of the more common questions.
If there are any "trouble spots" in your career path - like the fact that you failed a couple of years, or you changed your degree or career direction then think about what you would say to an interviewer about these: they will ask!
Remember when preparing your answers to these interview questions that the more specific you are the better. For instance, rather than saying: "I have excellent computer skills" say "I have used computers regularly since my first year at university and I am confident in Microsoft Word for Windows, Excel, and PowerPoint."

- Why did you decide to get this degree and attend this university?
- How will your degree help you to succeed?
- What would make someone successful in our business?
- What motivates you in a job and in your personal life?
- What role do you take ina group situation? Give examples.
- What research have you done on our firm and on our industry?
- How have you developed your interpersonal skills? Are they good?
- Describe your problem solving skills.
- Give me an example of a complex problem you solved.
- What are your short-term and long-term objectives?
- What do you look for in a job and why do you look for those things?
- Describe the perfect job.
- Why should I/we hire you?
- Can you work under pressure? Give an example.
- What type of salary are you worth, and why do you think so?
- What are the five biggest accomplishments of your life?
- How long would you stay with us?
- If you could change something in the course of your life, what would you change?
- What goals have you set for yourself? Why did you choose these?
- Describe your ideal picture of success.
- What interests you most about our position?
- What was the last book you read?
- Are you a leader? Give some examples.
- Describe your personality.
- What would you look for in hiring people? For this job?
- Describe the ideal employee.
- What were your responsibilities in your university activities?
- Describe your study habits.

- What other firms are you talking to and why?
- Why do you want to work in this industry, and for our company?
- What skills do you bring to us and how can you put them to work?
- What are your strengths?
- What are your weaknesses?
- I notice you failed your second year initially. Tell me about that?
- I see you took a year out of university. What did you do during that time?