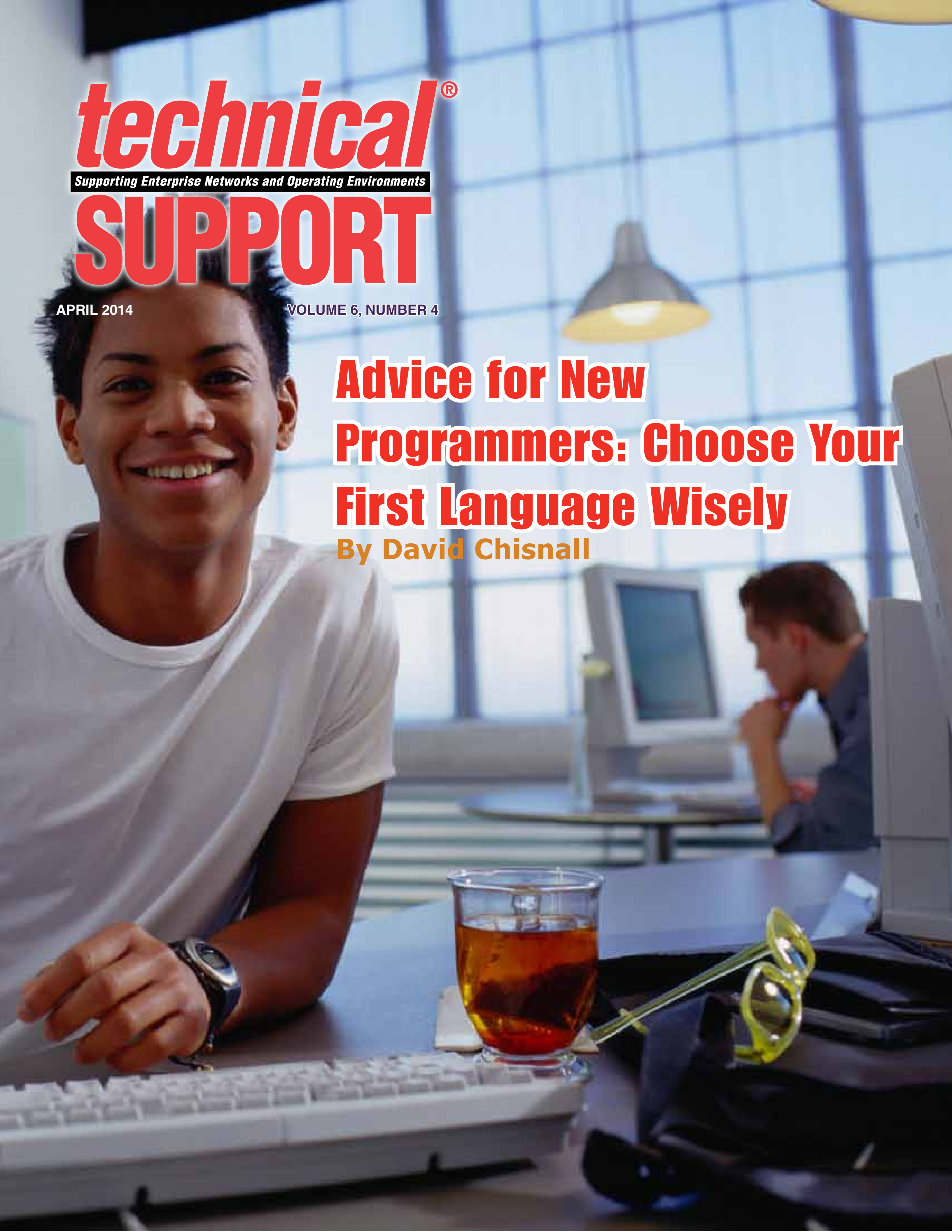
technical® SUPPORT

Supporting Enterprise Networks and Operating Environments

APRIL 2014 VOLUME 6, NUMBER 4

Advice for New Programmers: Choose Your First Language Wisely

By David Chisnall



Message from the President

Well, thank you again NaSPA - this time for helping me out with a family member! As many of you know I have a son, Leo III, that is recently out of the Navy. He is up to his neck in college and has just started the MIS and programming courses as part of his engineering curriculum. I got the inevitable question from him the other night.

"Dad, do you know anything about C++? This course is kicking my butt."

(Well, words to that effect anyway.)

Iraq and Afghanistan were not too tough for this guy but the concept of programming is a whole other thing. I remember being just as lost about thirty years ago, which brings me back to my point of THANK YOU NaSPA. As luck would have it, I only need to refer my son to this edition of NaSPA *Technical Support*. Indeed, I hope this latest edition helps many of our younger members as well with some great articles including, **The Components of a C Program** By Bradley L. Jones, Peter Aitken, and Dean Miller. Maybe even more on point**, Advice for New Programmers: Choose Your First Language Wisely** By David Chisnall. Many thanks to all of you as well as the other authors in this months' edition.

Now on to another topic: As you know NaSPA has a great Employment Site that is updated almost constantly. We are taking it to the next level. Our staff is in the process of integrating RSS feeds from our sources into the NaSPA web site at www.naspa.com. We will soon feature some of the choicest employment opportunities, automatically, right on the face of the NaSPA web site. This way you can check in whenever you like and instantly find the best I.T. jobs right there. We will also welcome employers to participate in this feed and get their positions out there in front of the finest I.T. professionals in the world, even before the job sites know about them.

If you are a NaSPA member, check the NaSPA web site often so you know when our new feed goes on line. If you are a potential employer, why not contact Jill Tucker right now on how you can skip non-productive means of filling job vacancies and get right to the people best suited to helping your organization grow. Her email is j.tucker@naspa.com. As always, feel free to email me at president@naspa.com and I'll make sure your communication gets in front of the right people.

Members and Employers, please consider NaSPA as a partner in your success. That's why we are here!

Best regards,

Leo A. Wrobel
Editor in Chief Technical Support Magazine
President, NaSPA, President@Naspa.com

technical SUPPORT
Supporting Enterprise Networks and Operating Environments

NaSPA

April 2014
Volume 6, Number 4

Published exclusively for NaSPA, Inc.
Network and Systems Professionals Association

www.NaSPA.com

Technical Support™ Magazine

Editor-in-Chief	Leo A. Wrobel president@NaSPA.com
Managing Editor	Sharon M. Wrobel sharon@b4ci.com
Associate Editor	Jim Justen jmjusten@gmail.com
Graphic Designer	Michelle Majerus-Uelmen graphics@techenterprises.net
Vice President of Staff Operations	Scott Sherer sherer@NaSPA.com
Membership Department	Nicole Cheever, 414-908-4945 Ext. 116, Manager NaSPA_membership@NaSPA.com
Advertising Sales	Jill Tucker, 414-908-4945 Ext. 111 j.tucker@NaSPA.com
CustomerCare Center	Bonnie Kuchinski, 414-908-4945 Ext. 450 customercare@NaSPA.com
List Rental	Jill Tucker, 414-908-4945 Ext. 111 j.tucker@NaSPA.com

7044 South 13th Street, Oak Creek, WI 53154
(414) 908-4945, (414) 768-8001 Fax

April 2014 Volume 6, Number 4

ARTICLES

4 **The Power of the Principles in Project Management**
By Jack Ferraro

6 **Advice for New Programmers: Choose Your First Language Wisely**
By David Chisnall

10 **Getting Your Disaster Recovery Plan Funded with an Awesome Business Impact Analysis: Part 1 of 3**
By Leo Wrobel and Sharon Wrobel

14 **Tapping the Quiet Power of Introverts in a Virtual World**
By Nancy Settle-Murphy

16 **Deployable Fiber Optic Systems for Harsh Industrial Environments**
By Rick Hobbs

19 **The Components of a C Program**
By Bradley L. Jones, Peter Aitken, Dean Miller

DEPARTMENTS

2 **President's Letter**

Call for Authors

Technical Support Magazine brings you an eclectic collection of articles, of interest to Information Technology professionals of all types. Do you have valuable insights and ideas that can be shared with NaSPA members? Fresh, timely ideas are important to our members, even if you have never published before. Our editorial staff is here to help and welcomes your submission. It's never too late to start. Contact president@naspa.com for more information or to submit your article for review for possible inclusion in a future edition of *Technical Support*.

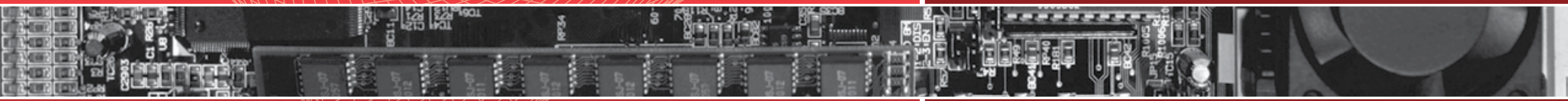
NaSPA Mission Statement:

The mission of NaSPA, Inc., a not-for-profit organization, shall be to serve as the means to enhance the status and promote the advancement of all network and systems professionals; nurture member's technical and managerial knowledge and skills; improve member's professional careers through the sharing and dispersing of technical information; promote the profession as a whole; further the understanding of the profession and foster understanding and respect for individuals within it; develop and improve educational standards; and assist in the continuing development of ethical standards for practitioners in the industry.

NaSPA serves Information Systems technical professionals working with z/OS, OS/390, MVS, VM, VSE, Windows Operating Systems, Unix, and Linux.

Join NaSPA now!

Call 414-908-4945, Ext. 116 or
e-mail NaSPA_membership@NaSPA.com
for more information.



The Power of the Principles in Project Management

By Jack Ferraro

In each project I take on for a client, I seem to relearn the same principles of success in project management. When I start to become curious about the latest project management fads—automated PMOs, Agile PM certifications, Lean Project Management—I find that I come back to these basic principles.

The first is that relationships are extremely important. Trust-based relationships make it possible to survive the chaos the change in organizations brings to people's lives. For me, the relationship with my customer—the person who is 50 percent of the provider/customer relationship that sustains the momentum of change—eventually becomes the critical make-or-break factor for me to be successful in serving them. This relationship involves an emotional investment by both parties, but it is ultimately based on a trust steeped in a level of respect for each other's competence and a shared vision. When my customers tell me to make tactical project decisions for them because they trust me, I know I have earned a special place in their eyes. When my customers take on the difficult political challenges that occur when organizations are trying to change, a task that is neither easy nor comfortable, they have in turn earned a unique spot in my professional life.

A second principle I relearn is that good planning is essential to project management. A good plan has the right amount of detail for the specific situation; it arrives in the right time to align stakeholders, and is easy to comprehend. I understand that a customers not accustomed to planning complex projects can become mentally frozen, not knowing where or how to begin to tackle the enormity of the initiative. Unfortunately, many unseasoned project managers hit the same icy mental state early in projects. I relearn that when I am overwhelmed with a mountain of project uncertainty, differing political agendas, and skeptical project team members, I have been able to refocus the negative energy on simply defining the real need and developing a mission statement to address that need. It sounds so simple but creates a platform to deal with some of the most difficult political issues early in the project, or at least have them acknowledged. This helps blunt personal agendas, sets a tone of serious discussion, and establishes how to measure project success. I relearn that staying focused on deliverables is essential and effective leadership, which sets direction and aligns resources to the mission. As soon as I stop focusing on deliverables, the project drifts into confusion, resources become unproductive, and morale plummets. Creating a portrait of the work of the project and delivering on that portrait creates believers in the project mission out of naysayers. The skill of articulating the story of the project—sequencing of the deliverables—and highlighting key decision points while instilling accountability in both the project team and customer team members works even in the most dysfunctional situations. Why? Because it is simple concept: a provider producing a deliverable for a customer. The basic premise is that the customer has the intelligence and competence to know what they want and the project team has the competence to produce it.

The skill of articulating the story of the project—sequencing of the deliverables—and highlighting key decision points while instilling accountablility in both the project team and customer team members works even in the most dysfunctional situations.

Everyone has participated in the model in some fashion during their life.

I relearn that strategic projects are embroiled in conflict—conflict between the status quo and agents of change, the risk averse and risk takers, the pacifiers and inciters. This conflict must be awakened for the organization to address and resolve the tension that has long been dormant. Without that tension, the organization is not likely to move forward. Managing this conflict is a part of the landscape. How you manage it will determine whether the conflict becomes positive energy to support lasting, fundamental change in the organization, or whether it becomes negative energy, worsening the organization's inability to change.

I relearn during each project that having healthy discussions about risk helps everyone sleep at night. The feeling of bearing the weight of serious project risk by yourself is not healthy. Having open dialogue about risks creates the reality of people being informed and sharing the responsibility of managing risk. Without accountability instilled through these basic planning competencies, risk is ignored or actively avoided at a cost to the organization and its employees—those same employees who come to work each day wanting to believe in the organization's leaders, management, values and mission.

I used to believe that only the project manager needed these planning skills. I was wrong. I have learned, and that learning has been reinforced in each project since, that these critical skills are required not just by the project manager—but by the customer. Take the opportunity to find a teaching moment to educate your customer.

However, no matter how good you are at building relationships and planning projects you still must be able to execute. Executing the project plan begins with communicating it to your stakeholders. I relearn each project how important it is to communicate and occasionally "sell" your plan. Sometimes the plan is so familiar and intuitive to me, I forget that stakeholders are seeing it for the first time. Those first few encounters with your key stakeholders are truly critical because the first impressions of your plan are lasting. They will leave with an impression that you "got it" or your "lost."

Jack Ferraro, PMP founded MyProjectAdvisor, a project management services company that provides project management training and leadership development. Jack has 20 years of experience working with project teams with extensive experience managing complex enterprise technology and business process improvement projects. A speaker and author, he can be heard at The PDU Podcast.

Advice for New Programmers: Choose Your First Language Wisely

By David Chisnall

Article is provided courtesy of Addison-Wesley Professional through NaSPA supporter Informit.com. Visit informit.com for more interesting content.

David Chisnall, who works on programming language design and implementation at the University of Cambridge, provides two tips for new programmers: choose your first language carefully, and take the time to learn the underlying theory behind it.

I learned to program when I was 7, on a BBC Model B with a dialect of BASIC and two dialects of Logo installed. The most important thing for me when I was starting was to find something interesting to work on. It's hard to be motivated to learn to do anything without some reason. I was taught to write some simple games (guess the number) and then learned about simple drawing and wrote small graphical games. Back then, the difference in quality between what I could create and commercial games wasn't too huge: most commercial games were written by one or two people and the limitations of the machine were such that you didn't get significantly more complexity by adding more people.

There are a few potential pitfalls when you're learning to program on your own. In particular, it's very easy to pick up bad habits, and the more that you practice them the worse they get. This is most apparent if you try learning a language that is designed for industrial use first, rather than one designed for teaching. This is part of the reason why I advocate trying lots of different programming languages. Languages like Python make terrible first languages. Anything that describes itself as "multi-paradigm" generally means that it badly implements a lot of possible programming models.

If you want to learn object-oriented programming (and you almost certainly do), then the best place to start is Smalltalk. This, in a system like Squeak or Pharo, gives you an environment where you can inspect everything and where everything is an object. The language is simple enough that it takes half an hour or so to learn, but the time you spend trying it will teach you to write good object-oriented code in any language, even if you never touch Smalltalk again. In contrast, learning a language like Java or C++ first will give you a myriad of bad habits that are very hard to break.

I'd also strongly recommend learning an assembly language, or at the very least a low-level language like C. Again, you don't have to ever use them later, but understanding how high-level constructs map to something that the computer can execute is incredibly valuable.

Understanding the Underlying Theory of Languages

BBC BASIC was not a bad first language, in spite of the overwhelming prejudice against BASIC. It included an inline assembler, so you could get a feel for exactly how things executed on the 6502 processor in the machine, and let you directly manipulate memory via PEEK and POKE, typically for interacting with memory-mapped I/O. Unlike many contemporary dialects of BASIC, it supported structured programming, via subroutines.

I probably wouldn't recommend the BBC micro to new programmers today, because the sparsity of the development environment would be intimidating, but it's worth remembering that the first language that you learn almost certainly won't be the one that you use for real projects later. If it is, you're probably doing something badly wrong, because you won't ever fully understand the limitations of one language until you've learned a few more. There's a small chance that after learning a dozen languages you find the first one you learned was the best for everything that you need to do, but it's quite unlikely.

One trap that self-taught programmers often fall into is neglecting the importance of the underlying theory.

I began learning to program when I was 7, and when I arrived at university I was pretty sure that I already knew all of the programming-related parts of the course. It turned out that my lack of knowledge of graph theory and complexity theory was a serious limitation.

In general, the first year of a computer science program tries to teach you two fundamental concepts. If you understand these two ideas well, then you should have no problem with the rest of the course. If you don't, then you'll struggle. These concepts are induction (recursion) and indirection (pointers).

Induction

Induction is the idea that you can define an infinite series by defining a few simple cases and defining a rule that allows you to reduce a complex sequence to a simpler one. For example, you can define multiplication in terms of addition as:

1) n × 0 = 0
2) n × 1 = n
3) n × m = n × (m - 1) + n

If you then want to evaluate 5 × 3, then you look for the first matching rule. It's the third, one, which gives: 5 × (3-1) + 5, or 5 × 2 + 5. Apply it again to that and you get 5 × (2-1) + 5 + 5, or 5 × 1 + 10. Now, the second rule applies and so the 5 × 1 part becomes 5, and so the result is 15.

This is very powerful, because it's simple pattern matching and then applying rules: something that the computer is very good at. You could implement this in a programming language as something like:

```
int mult(int x, int y)
{
 if (y == 0)
   return 0;
 if (y == 1)
   return x;
 return mult(x, y-1) + x;
}
```

Multiplication is a pretty trivial example: most computers have hardware for doing multiplication, and it's a lot faster than using this approach, but for more complex things, if you can understand the problem in terms of induction then you can implement it in terms of recursion. This is one of the reason why most universities teach Prolog to undergraduates: you can't write even simple programs in Prolog unless you understand induction, and once you understand induction you've got a very powerful tool to reason with.

Indirection

The other core concept, indirection, is fundamental to building complex data structures. It's simply the idea that a variable, rather than containing a value, can tell you where to look for a value. In a computer, your data is stored in memory and so a variable name is just a human representation of the address of something in the computer's memory. A pointer is just the address of another bit of memory, stored in memory.

Languages like C allow arbitrary levels of indirection, so you can have pointers to pointers to pointers and so on. To the computer, it's nothing special. Most computers don't distinguish between data and addresses (which has been the source of myriad security vulnerabilities over the years, but that's another story), so you can store an address anywhere you can store data. The computer just loads the value and, if you treat it as an address rather than data, and loads the value from that address.

Some IBM mainframes tried to optimise for this by having special values that, when you loaded them, would actually load the result. This caused some problems when people created circular sequences of these

Insurance savings start here.

Liberty Mutual.

Give Your Information Technology Career That "Extra Edge."

Get Published with NaSPA

The **N**etwork **a**nd **S**ystems **P**rofessionals **A**ssociation (NaSPA) has provided award winning publications like *Technical Support Magazine* to Information Technology (IT) professionals worldwide since 1986. Our publications begin and end with people like YOU. NaSPA has published the work of hundreds of IT professionals for the benefit of thousands of members and IT practitioners. Do you have what it takes to be a NaSPA contributor? We invite you to find out today.

Call for Original Articles and Technical Papers

NaSPA invites you to submit a 1000-1500 word article or research paper for one of our publications. The topic is open as long as it is germane to IT, however mainframe, data center, and other high end operations and systems programming subjects will be given preference.

You must be a NaSPA Member in good standing. *Gold Membership is FREE.* Platinum Members enjoy higher publication priority for only $39.00 yr. All members receive:

- ***Technical Support*** **Magazine Subscription**
- **Employer Advertising Discounts**
- **Expansive On-Line Technical Library**
- **Job Placement and Hiring Site**
- **Discounts on NASTEC and IDCE Shows**
- **Discounts on Auerbach and Cisco Press Publications**
- **Home and Car Insurance Discounts**
- **More**

NaSPA®

Restrictions apply and all articles become property of NaSPA. For more info and author agreement Email president@naspa.com. First time authors welcome. Remember, whether you are already in the IT profession or are still in school, articles look GREAT on a resume! Corporate submissions acceptable if they are not overt sales pitches. **What are you waiting for? Get your name out there today in one of NaSPA's great publications!**

values, causing the computer to loop forever trying to find the result that wasn't just another pointer.

Ignore Theory at Your Own Risk

Pointers are one of those things that are hard to explain, because once you understand them you wonder how you (or anyone else) ever found them complicated, but they're one of the core building blocks of programs. One of the problems with using languages like Java or Python for teaching is that students only learn about references, which are a somewhat reduced version of a pointer, lacking the generality in the interests of ease of use. While Java references are easier to use correctly than C pointers, they're slightly harder to understand, because now you have some things that can have references to them (objects) and lots of other things that can't (references, integers, floating point values), and no obvious reason why not. In contrast, C lets you use the full expressiveness of the underlying machine, even if you then use that to shoot your own feet off.

David Chisnall arrived at the Swansea University in 2000, looked at the sun and sea, and decided to stay there. Three years and one degree later, he was no longer under the illusion that the sun was a regular feature, but was persuaded to remain for another degree by the promise of a desk with a view of the sea. During his time as a PhD student, he worked hard at the best known of postgraduate activities: procrastination. This involved writing portions of A Practical Guide to RedHat Linux, Second Edition and regular articles for InformIT and a local tech news startup (which, as these are prone to do, has since gone bust).

David is an active member of the open source community. He is a founding member and core developer of the Étoilé project, which aims to build an open source user environment based for desktop and mobile computing systems on top of GNUstep. He also contributes to GNUstep and is the author of the GNUstep Objective-C runtime and maintains Objective-C support for open source operating systems in LLVM/Clang. In 2012 he was elected to the FreeBSD Core Team. His contributions to FreeBSD include improvements to locale support in libc, the port of libc++, and a replacement C++ runtime library: libcxxrt.

In 2007, David's first book, The Definitive Guide to the Xen Hypervisor, was published. This was begun as a procrastination activity, to distract himself from his looming PhD thesis deadline, and was successful: the book and the thesis were both completed within a fortnight of each other. He spent the next few years working freelance and writing three more books, two about Objective-C and one about Go, before returning to academia. He is now part of the Security Group at the University of Cambridge Computer Laboratory where he works on language and hardware co-design and continues to consult on compiler and language-related topics.

COMPETITIVE PRICING + ACCOUNT MANAGERS WHO WORK WITH YOU TO FIND THE BEST SOLUTION---

WWW.SERVERRACKOPTIONS.COM
800-458-6255

SERVER RACKS - STANDARD - CUSTOM

PDU'S - KVM SWITCHES - PORTABLE A/C RACKMOUNT KEYBOARD MONITORS AND MUCH MORE!

SEE OUR NEW AUTO-LOCK C13 POWER CORDS & JUMPERS

ISC™ Information Support Concepts, INC
Leading Information Technology Products Since 1987 ™

Advance Your Career Today!
CISCO.
SAVE 30% on all Cisco Press Certification and Networking Technology learning products today.
Shop Now At ciscopress.com
Use Coupon Code TECHSUPPORT
Cisco Press

Coupon TECHSUPPORT valid for 30% off the list price of eligible products purchased on ciscopress.com. Coupon may not be combined with any other offer and is not redeemable for cash. Coupon not valid on book + eBook bundles, Rough Cuts, Safari Books Online, or Live Online Training. Coupon expires on December 31, 2013. Offer subject to change.

Getting Your Disaster Recovery Plan Funded with an Awesome Business Impact Analysis: Part 1 of 3

By Leo Wrobel and Sharon Wrobel

In the first of a three-part series, Leo A. Wrobel and Sharon M. Wrobel discuss some of the crazy ideas that you may have about why management won't fund your disaster recovery plan. You need to revise your thinking to create a successful business impact analysis that will put money in your budget.

There's an old saying I've often quoted: "Lawyers use figures the way a drunk uses a lamppost—for support, rather than illumination." It's kind of like that when you decide to take the plunge and ask management to fund a disaster recovery plan. The figures you use and the presentation you make can support your plan and illuminate management—but only if you do it correctly. If you put together a compelling business impact analysis (BIA) and conclusively convince management of the vulnerability that exists in the organization's environment, your recovery plan WILL be funded.

"Lawyers use figures the way a drunk uses a lamppost - for support, rather than illumination."

That statement brings to mind a routine from Larry Fine of "Three Stooges" fame:

Rich Guy: "If you had a dollar, and your father gave you another dollar, how many dollars would you have?"

Larry: "One dollar."

Rich Guy: "You don't know your arithmetic!"

Larry: "You don't know my father!"

If you substitute the word boss for father, it probably closely approximates the thought that went through your mind the instant that I said your plan would be funded: "Leo, you don't know my boss!"

Actually, I do know your boss. Probably not literally, of course, but I have worked with the animal before. Nobody is more acutely aware than I am that money is very hard to come by these days—after all, I work as a consultant. Even so, I'm not starving in this profession (well, most months, anyway). I like to think that this is because, after 25 years in the disaster recovery business, I know a thing or two about convincing skeptical management.

In this three-part series, I'll impart a few tricks of the trade that might help you to finally get some funding and commitment for your recovery plan. It's not as tough as you may think, provided that you do your homework in advance.

Let's start by dispelling a few myths.

Myth 1: Management Doesn't Care About Disaster Recovery

Wrong, wrong, wrong. Disaster recovery, contingency planning, risk mitigation, or whatever you want to call it is a fundamental and fiduciary responsibility of executive management. The topic often comes up at Board of Directors meetings, and right on down the totem pole. I know from experience, because I've been a CEO for a corporation.

By its makeup, a corporation can do anything a person can do—except commit a crime. If that were to happen, the people who lost equity in the corporation due to my (in this example) gross negligence would not stop at suing the corporation—they could come after me as well. Why? Because they can.

This fact is never far from management's consciousness, particularly since passage of the Sarbanes-Oxley Act of 2002, which provides a whole bunch of ways in which CEOs and CFOs can be held accountable for things. In fact, you will be every bit as hard-pressed to find an executive who says a disaster recovery plan is a bad idea as to find a person who says a pre-need funeral plan is a bad idea. Most people agree that a pre-need funeral plan—in principal, at least—is a good idea. Having said that, do you have one? How many people do you know who do have one? Answer: Not many.

The same problem exists with recovery plans, which are delayed for many of the same reasons, centering around competing priorities, lack of time, or a sense that it will be too expensive. All of these kinds of concerns can be assuaged through a compelling BIA and crisp presentation that puts the problems in terms the executive can understand.

So what can you expect from your BIA and resulting presentation? I've mentioned in lectures and in previous articles that management can respond with one of three answers to a funding request:

Yes.

No.

Let's study this some more.

Which of the three do you suppose is the most common answer? If you chose "Let's study this some more," you're correct, and no doubt have been through such a request process.

So what can you do to get a "Yes" response? I like to stack the deck before going into a meeting to request funding. I'll show you some of those tricks in this series, but first let's continue myth-busting with another common misconception:

Myth 2: Management Doesn't Understand a Disaster's Impact on the Business

Wrong again! Management has a very good idea of the impact of a disaster on the business. Think about it. Senior executives get paid based on how the business performs (in theory, anyway). Vice presidents of marketing or sales, business controllers, etc. know where the money comes from; it's part of their scope of responsibility and probably part of their compensation plan. So it's possible to play on this fact a little. If you're successful in showing an executive the impact of a disaster on the business, in terms that he or she understands (read: dollars), you'll have the exec's attention—and, more importantly, gain his or her support.

Obviously, one of the first things you'll need is a crisply presented business impact analysis. With this analysis, using non-technical terms that management can understand, your plan will be funded. If there are any ambiguities or doubts, however, management will want to "study some more," and you could walk out of your meeting with a longer to-do list than you had when you went in.

Myth 3: Management Will Never Fund a Recovery Plan

It has been my experience that people who strike out on funding year after year have something very wrong with how they're asking for money. Everyone will agree that the last thing management wants to endorse is money down a rat hole. If you don't have your facts straight, that's just how your funding request can be taken. If you can apply a proposed expenditure to a problem that everyone in the room agrees needs to be addressed, your plan will be funded. Executives are not in those seats because they're bashful, and under the circumstances they will spend money.

So how do you convince management of the legitimacy of your request? What kinds of facts are most relevant and important to management?

Fundamentally, management needs to know only four things in order to decide whether to fund your plan:

- What can happen? (Fire, flood, hurricane, sabotage, etc.)
- What is the probability that it will happen? (Expressed best in percent probability of the event in a given year.)
- What does it cost when it happens? (Think in terms of lost sales, market share, employee productivity, and customer confidence.)
- What does preventing it cost? (Present a high-level overview of the proposed protective system, procedure, or function.)

There is a possible fifth question you can also address: What are the other factors? (Consider legal liability, government requirements, Sarbanes-Oxley, etc.)

What Should Be Included in a Business Impact Analysis?

First, consider that your company probably doesn't do only one thing. It has different businesses, or at a minimum different business dynamics in each business unit. All have different pain thresholds. Some can last a month after a disaster, others scream within hours. If you don't take this factor into account, management will discount your BIA.

Your Organization. It is What it Eats.

Supplement Facts

Serving Size: All You Can Eat
Servings Per Container: 1

Amount Per Serving	% Daily Value
Energy	100%
Experience	100%
Enthusiasm	100%
Initiative	100%
Technical Savvy	100%
Reliability	100%

* Percent NaSPA-Established Daily Values based on organizations that want to succeed and prosper!

Are YOU Hiring? Find one of the most useful EMPLOYMENT SITES in the industry, proudly sponsored by one of the most trusted names in information technology.

Since 1986, NaSPA (the Network and Systems Professionals Association) has been the premier not-for-profit advocate to Information Technology (I.T.) and Network professionals worldwide. Thousands have coursed through NaSPA training programs, subscribed to its award winning publications, attended its conferences and trade shows, and enjoyed the many benefits of membership. Platinum members, access our expansive SOFTWARE LIBRARY including recent contributions and those hard to find "legacy" applications!

If you are looking for the perfect employment candidate, it's all in one place you can *truly trust.* Explore the many benefits NaSPA offers to its members, for FREE. At the same time re-energize your organization with proven nutrition from our vast membership pool. Don't do anything until you talk to us! Dollar for dollar we are the best value in the industry for finding that next candidate. Visit our EMPLOYMENT SITE and see for yourself, or email Jill Tucker at j.tucker@naspa.com.

NaSPA®
NETWORK AND SYSTEMS PROFESSIONALS ASSOCIATION
www.naspa.com

7044 S. 13th St.
Oak Creek, Wisconsin 53154
(414) 908-4945 x 111

The key to selling to management is learning how to communicate in management's terms.

Note

Screaming is not the litmus test of being mission-critical. Often, the people who scream loudest are the least mission-critical, while others who don't make a sound can cripple the company if they're affected.

Next, you'll need to learn about each business unit, including its pain threshold, to present a valid analysis. Talk to responsible managers and executives that management holds in confidence. That way, when the time comes for the big funding request meeting, management will have seen these figures. How can management disagree with its own figures? See what I mean by stacking the deck beforehand?

Obviously, I have no idea what kind of company you work for. But I can make a few assumptions for you, and the next article in this series will walk through a sample BIA. I'll even include slides from winning presentations for real companies. Produced and presented properly, these slides can be the cornerstone of your executive presentation for support and funding. They also serve as a springboard to fruitful discussion and thoughtful technological planning.

As a litmus test, we'll ask whether your non-technical wife, husband, father, mother, or grandmother would understand the messages your slides carry. If so, your presentation is ready for management. Don't laugh! This kind of approach in making presentations to executives can be used in a wide variety of companies, and for purposes that go beyond disaster recovery. If you can "sell" management on a disaster recovery project, you can sell them on other things you need. The key is learning how to communicate in management's terms. They're not going to learn yours! And if you wait year after year for them to understand you, all you'll get is frustration—not money.

See you next month!

This series was previously published by longtime NaSPA supporter Informit.com.

NaSPA President Leo A. Wrobel has more than 30 years of experience with a host of firms engaged in banking, manufacturing, telecommunications services, and government. An active author and technical futurist, he has published 12 books and more than 1000 trade articles on a wide variety of technical subjects. Leo served 10 years as an elected mayor and city councilman (but says he is better now). A sought-after speaker, he has lectured throughout the United States and overseas and has appeared on several television news programs. Leo is presently CEO of Dallas-based TelLAWCom Labs Inc. and b4Ci, Inc. Contact Leo at 214-888-1300 or email leo@b4ci.com.

Sharon M. (Ford) Wrobel is a Director at NaSPA and Managing Editor for *Technical Support* Magazine. She can be reached at sharon@b4ci.com.

SecurityRecruiter.com

P.O. Box 398
WOODLAND PARK, CO 80866
PHONE: 719-686-8810
PHONE: 877-417-6830

Tapping the Quiet Power of Introverts in a Virtual World

By Nancy Settle-Murphy

Think about it: There's zero correlation between being the best talker and having the best ideas. And yet, according to Susan Cain, author of the groundbreaking book, Quiet: The Power of Introverts in a World That Can't Stop Talking, our society is overwhelmingly biased toward extroverts.

This bias is glaringly obvious in our workplace, where teamwork, groupthink and collaboration are prized over deliberate, solitary work and quiet thought. In a world where the "ideal" employee is seen as gregarious, action-oriented, decisive, confident and bold, introverts are often undervalued, overlooked and dismissed. The kind of "people skills" that extroverts often exhibit are increasingly emphasized in performance reviews, while qualities more likely associated with introverts, like reflection, thoughtfulness, and quiet listening are rarely mentioned.

This favoring of extroverts is especially true in the virtual workplace, where introverts are often bypassed in favor of their more garrulous, dynamic colleagues. Why? For pretty much the same reason the proverbial squeaky wheel gets the grease: People who are quick to volunteer their ideas and freely share opinions can make life a lot easier for virtual leaders who otherwise must figure out ways to cajole and persuade their more introverted counterparts to join in the conversation. Far less effort to let the more extroverted team members lead the conversations, hoping they may somehow inspire the more reticent ones to participate—eventually.

In this article, I explore ways that virtual team leaders can learn how to take advantage of the quiet power and special strengths of the introverts on their teams, instead of trying to make their introverts conform to the "extrovert ideal." (Please note I have made some generalizations about introverts and extroverts for instructive purposes. In reality, many of us, at least from time to time, display qualities of both extroverts and introverts, depending on the situation.)

Remarkable powers of concentration: Introverts have the ability to focus on problems, especially those that require deep thinking, for unusually longer periods of time, especially if they have are freed up from having to work with colleagues in the name of "teamwork." As a team leader, consider assigning your introverts those tasks that can be done well (if not better) independently, with relatively few dependencies on other team members to get the job done. Ideal projects might include gathering primary or secondary data, analyzing results, assimilating recommendations, and formulating action plans.

Ability to articulate clear ideas in writing: Because introverts tend to stop, reflect and absorb information before speaking, they reflect this kind of deliberate thought in their writing as well. Need someone to formulate a cogent proposal, recommendation, summary or other presentation of important ideas? Your best bet

just may be an introvert on your team. (This may be counterintuitive to some leaders, who often assume that charismatic speakers can successfully translate those same speaking skills into written form. The opposite is often true!)

Active listening and diagnostic abilities: Introverts tend to be more comfortable listening, versus speaking. This is an especially crucial skill in the virtual world, where acute listening skills help us to fill in the gaps left by the absence of visual cues. Tap into the introvert's exceptional ability to listen by asking him or her to formulate insightful questions, conduct important interviews, or act as a note-taker or observer for team meetings, paying close attention to invisible organizational dynamics that be hard for others to discern. Introverts do a particularly good job demonstrating their sharp listening skills by paraphrasing and summarizing key points, whether verbally or in writing.

Deep reflection: To make their best contributions, introverts need time to pause, reflect, and think before speaking. That's why it's vital to design your team meetings, whether in person or virtual, to build in time to allow people to pause and think, whether by using silence to allow time for reflection, or providing a quiet forum for participation, such as by using an online conference area, or asking people to jot down ideas quietly. This way, you enable introverts to contribute their ideas and opinions more easily and freely, and you also help to balance participation between your extroverts, who can sometimes be a tad overzealous in their verbal contributions, sometimes causing your more circumspect counterparts to withdraw or shut down.

Attention to detail: Enable your introverts to participate fully in your next important team meeting by clarifying objectives, stating goals, and setting expectations up front. Make sure everyone knows what content will be important to learn and assimilate in advance, and what questions or ideas they should have ready to bring to the table. (If you're thinking that designing a pre-work component to your meeting is an unnecessary bother since most people don't do it – reconsider your assumptions. While it may be true that not everyone will invest time and thought in planning how they want to participate ahead of time, you can bet that your introverts will appreciate the opportunity.)

Relationship-building. While your extroverts socialize more easily with many different kinds of people, your introverts have a particular talent for developing close, trusting relationships with those who are truly important to them. (Many people assume that an introvert is overly shy just because s/he is selective about with whom to socialize. It's more often the case that the introvert has less tolerance for social chit-chat and superficial relationships in favor of more meaningful discussions and deeper connections.) If your team needs stronger connections with key people in other organizations, or perhaps with your clients, the introverts on your team can be your best choice. While your social butterflies may draw energy from relating to sheer numbers of people, your introverts are more switched on by fewer, deeper connections—the kind that build lasting relationships.

Creative super-powers. Still think that the best ideas are generated by groups of people working together in real-time? Many recent studies refute the notion that group brainstorming necessarily results in the highest quantity or quality of innovative ideas. In fact, it is often independent brainstorming that yields the best ideas in the shortest period of time, second only to online brainstorming, where people can contribute ideas in a shared area at any time, from anywhere. Some people may be at their creative best when they can bounce ideas off of each other in real-time, while introverts tend to do their best innovative thinking alone. Leaders who rely on their team members to ignite that creative spark must find ways to enable all kinds of brainstorming, in different venues, at different times.

In a world that correlates the strength of an organization's teamwork to its overall success, the real value of introverts often gets overlooked. In the virtual world, it's even easier to ignore or dismiss introverts as being key members of the team, simply because it can take so much more effort to find ways to enable them to make their best contributions. Assuming that your team represents a microcosm of the larger world where fully one-third to one-half of all people are introverts, it's time to find new ways to leverage the potential of some of your quietest team members, who just may be the best and the brightest, if only you take time to hear them.

Nancy Settle-Murphy, Guided Insights founder and principal consultant, draws on an eclectic and varied combination of skills and expertise. She wears many hats, depending on the challenges she is helping clients to solve. She acts as meeting facilitator, virtual collaboration coach, change management leader, workshop designer, cross-cultural trainer, communications strategist and organizational development consultant.

http://www.guidedinsights.com/about-us/

Deployable Fiber Optic Systems for Harsh Industrial Environments

Deployable systems, often exposed in harsh industrial environments, come battle-tested

By Rick Hobbs

As the utilization of fiber optics has increased within the industrial sector, so have the number of "deployable" systems used in applications from oil and gas exploration, drilling and distribution to mining.

As opposed to fixed installations, deployable systems are designed to be quickly installed, retracted, and then relocated in the field and even deep underground in some of the most inhospitable environments on earth.

Given the environments in which they reside, industrial-grade fiber optic systems are typically commercialized versions of field-tested, proven military-grade products.

As such, the component parts of the system are designed to withstand everything from dust and debris to chemical exposure, temperature extremes, UV, radiation, electrical power transients, interference, fire, moisture, humidity, water, crush, tension, flexing, impact, and vibration.

Rick Hobbs, Director of Business Development at Optical Cable Corporation (OCC), explains that when designing a deployable fiber optic system, it needs to be looked at in its entirety. Unlike fixed applications, a deployable system is designed from beginning to end (plug and play) and delivered to the customer as a complete solution. OCC designs and manufactures fiber optic cable, connectors and assembly solutions for harsh and rugged environments.

According to Hobbs, the primary elements of a deployable system include hardened cable jacketing; "genderless" connectors for quick deployment without regard for male or female ends; hybrid systems that include copper along with fiber to deliver data communications and power; and reel systems that speed deployment and retraction while protecting the fiber while not in use, or during transit.

Hardened Cabling

For purposes of deployment, OCC typically recommends its tight bound, tight buffered distribution style cabling, which is ideal because of its small diameter and lightweight construction.

Distribution-style cables have a tight-bound outer jacket, which is pressure extruded directly over the cable's core. This combination of a helically stranded core, and a pressure extruded outer jacket provides an overall cable construction that offers better crush and impact protection and increased tensile strength. This also reduces outer jacket buckling during deployment.

According to Hobbs, escalating degrees of cable protection are available as needed to meet the specific needs of an application.

Various jacket materials are available, including PVC or polyurethanes, which are specifically tailored to meet the mechanical and environmental needs of the application. Options within each jacket material include coefficient of friction, cold temperature flexibility and temperature range, to name a few.

Water tolerant options are available that take advantage of the qualities of tight buffered cable and super absorbent polymer aramid yarn.

Fiberglass or metal braided jackets not only provide excellent abrasion resistance, but also deliver increased rodent protection. Custom rodent resistant cables are available that include metal or dielectric armor or additives to the outer jacket.

"In deployable applications, exposed cable is often an intriguing temptation for animals, which can, and often do, chew on it." says Hobbs.

Hybrid Cables, Connectors

For applications that can benefit from fiber optics and copper, hybrid connector-cables offer both within the same cabling sheath.

A distinct advantage of hybrid cable-connector solutions is that the customer can bundle both the high performance of fiber with the copper power or control signals in one cable. This reduces the number of cables that must be designed, purchased and deployed into a system.

It also offers distinct savings in labor and cable structure costs for the customer.

Genderless Connectors

"Genderless" connectors have both male and female elements, and perhaps are more appropriately described as dual-gender. They are designed for quick deployment, allowing the user to unreel fiber cable without regard for male or female ends.

Companies such as OCC have further simplified the genderless design with user friendly mating interfaces (the company's EZ-Mate family) capable of "blind mate" and/or applications that require thousands of mating cycles.

In addition, the connector system is designed to resist extreme harsh mechanical and environmental conditions including high vibration, mechanical and thermal shock, and fluid immersion.

Another benefit of genderless connectors is that multiple identical cable assemblies can be daisy-chained (sequenced) together to extend the distance of a deployable system while maintaining polarity. Polarity can be an issue when connecting an odd number of traditional male to female gender connectors. In such cases, an additional connector is required to correct polarity. However, such connectors are known for high loss and add additional components for the customer. Therefore, genderless connectors are uniquely advantaged over traditional interconnection systems.

Distances of several kilometers are possible, limited only by system link budget (dBm).

"This type of genderless connector provides extreme flexibility in the case of redeployment, where the length of the cable assemblies required for the next application are not fixed, or even known," says Hobbs.

Reel Systems

The key characteristics of a reel system in deployable fiber optic applications are that it is lightweight and stackable for storage and transit, says Hobbs.

To meet these requirements, companies such as OCC are providing lightweight alternatives to traditional metal reels. Constructed of durable, yet lightweight, impact absorbing polymers, these modular advanced reel systems (MARS®) are designed specifically for the demanding needs of harsh-environment fiber optic installations.

Reels can be used with simple deployable axle or a flange supported deployment and acquisition system. These types of systems include A-Frames, cable acquisition cradles, transit case systems, tripods, bumper mounts, backpacks, backpacks with fiber optic slip rings, and cartridge systems.

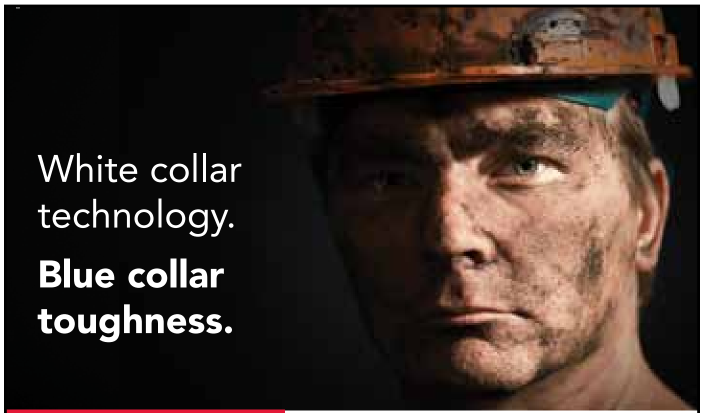
The cartridge system, which comes with casters, is an ideal choice in many deployable applications.

"Using a cartridge system, a single person can handle multiple spools at once and can quickly deploy fiber and rewind on the reel without assistance," says Hobbs.

To simplify shipping and transit, cartridge systems, transit cases and reels are designed with interlocking stacking features.

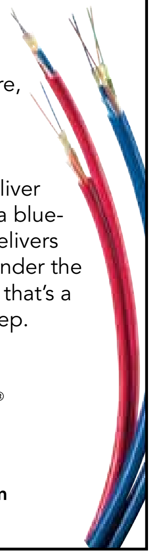
Reel systems also provide a measure of protection of fiber optic cabling for unspooled cabling, or when the cabling is retracted.

"In harsh environments, when you can put your fiber optic assemblies in a controlled environment storage system like a reel, any potential damage to the cable or the connectors is minimized," says Hobbs. "This reduces the need to refurbish components regularly, because it the system better protected during its deployment."



Count on OCC for:
- Security
- Environmental Monitoring
- Voice/Data/Video Communications
- Automation
- Production Rates and Yields
- Equipment Control and Monitoring
- ERP

With cables and connectivity developed to resist chemical exposure, moisture, dust, extreme temperatures, and other conditions often found in mining operations, we deliver a high-tech product with a blue-collar work ethic. OCC delivers quality communications under the toughest conditions. And that's a commitment that runs deep.



OCC®
OPTICAL CABLE CORPORATION
800-622-7711 | occfiber.com

Wireless Access/Data Communications

Although deployable fiber optic systems are largely “wired,” hybrid cabling (the combination of fiber optic and copper/electrical within the same cable sheath) also allows for installation of wireless access points anywhere, even underground. This is ideal when access points are constantly changing.

Unlike traditional wireless networking devices that require 110-Volt AC power for each device, with a hybrid system power can be supplied in the same cable that also carries voice and data.

As a result, any 802.11-certified devices are able to communicate through the network, including personal devices such as PDAs, laptops, VOIP devices and cell phones.

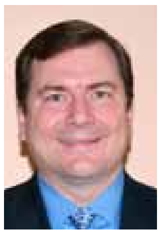
This provides personnel even deep within mines with the means to communicate with each other and even make calls outside the system. In addition, sensor-based data such as temperature, humidity, airflow and gas can also be collected and delivered wirelessly for use by the entire network.

Increasing Conversion to Fiber Optics

According to Hobbs, there are many industrial companies that are converting to fiber optics as the costs for components continue to drop, making fiber a better solution than copper in most applications. Even die-hard copper devotees are moving to fiber and when they do, they rarely look back.

“When System Engineers realize the bandwidth opportunities, they usually expand their capabilities, and identify creative new ways to enhance the solutions for their applications,” says Hobbs.

For more information about deployable fiber optic system for harsh environments, contact Optical Cable Corporation (OCC) at 5290 Concourse Drive, Roanoke, Virginia, 24019; Phone: (800) 622-7711, Canada (800) 443-5262; FAX: 540-265-0724; Email: info@occfiber.com; Visit the web site www.occfiber.com.



The Components of a C Program

By Bradley L. Jones, Peter Aitken, Dean Miller

Sample Chapter is provided courtesy of Sams Publishing, an affiliate of longtime NaSPA supporter Informit.com. NaSPA thanks informit.com for their contributions and recommends that NaSPA members visit informit.com for more insightful content.

In this lesson you will learn the components of a short C program, the purpose of each program component, and how to compile and run a sample program.

Every C program consists of several components combined in a specific way. Most of this book is devoted to explaining these various program components and how you use them. To help illustrate the overall picture, you should begin by reviewing a complete (though small) C program with all its components identified. In this lesson you learn:

- The components of a short C program
- The purpose of each program component
- How to compile and run a sample program
- A Short C Program

Listing 2.1 presents the source code for bigyear.c. This is a simple program. All it does is accept a year of birth entered from the keyboard and calculate what year a person turns a specific age. At this stage, don't worry about understanding the details of how the program works. The point is for you to gain some familiarity with the parts of a C program so that you can better understand the listings presented later in this book.

Before looking at the sample program, you need to know what a function is because functions are central to C programming. A function is an independent section of program code that performs a certain task and has been assigned a name. By referencing a function's name, your program can execute the code in the function. The program also can send information, called arguments, to the function, and the function can return information to the main part of the program. The two types of C functions are library functions, which are a part of the C compiler package, and user-defined functions, which you, the programmer, create. You learn about both types of functions in this book.

Note that, as with all the listings in this book, the line numbers in Listing 2.1 are not part of the program. They are included only for identification purposes, so don't type them.

Input down-arrow.jpg

Listing 2.1 bigyear.c - A Program Calculates What Year a Person Turns a Specific Age

```
1:  /* Program to calculate what year someone will turn a specific age */
2:  #include <stdio.h>
3:  #define TARGET_AGE 88
4:
5:  int year1, year2;
6:
7:  int calcYear(int year1);
8:
9:  int main(void)
10:  {
11:     // Ask the user for the birth year
12:     printf("What year was the subject born? ");
13:     printf("Enter as a 4-digit year (YYYY): ");
14:     scanf(" %d", &year1);
15:
16:     // Calculate the future year and display it
17:     year2 = calcYear(year1);
18:
19:       printf("Someone born in %d will be %d in %d.",
20:             year1, TARGET_AGE, year2);
21:
22:     return 0;
23:  }
24:
25:  /* The function to get the future year */
26:  int calcYear(int year1)
27:  {
```

```
28:     return(year1+TARGET_AGE);
29:  }
```

Output down-arrow.jpg

```
What year was the subject born? 1963
Someone born in 1963 will be 88 in 2051.
```

The Program's Components

The following sections describe the various components of the preceding sample program. Line numbers are included so that you can easily identify the program parts discussed.

The main() Function (Lines 9 Through 23)

The only component required in every executable C program is the main() function. In its simplest form, the main() function consists of the name main followed by a pair of parentheses containing the word void ((void)) and a pair of braces ({}). You can leave the word void out and the program still works with most compilers. The ANSI Standard states that you should include the word void so that you know there is nothing sent to the main function.

Within the braces are statements that make up the main body of the program. Under normal circumstances, program execution starts at the first statement in main() and terminates at the last statement in main(). Per the ANSI Standard, the only statement that you need to include in this example is the return statement on line 22.

The #include and #define Directives (Lines 2 and 3)

The #include directive instructs the C compiler to add the contents of an include file into your program during compilation. An include file is a separate disk file that contains information that can be used by your program or the compiler. Several of these files (sometimes called header files) are supplied with your compiler. You rarely need to modify the information in these files; that's why they're kept separate from your source code. Include files should all have an .h extension (for example, stdio.h).

You use the #include directive to instruct the compiler to add a specific include file to your program during compilation. In Listing 2.1, the #include directive is interpreted to mean "Add the contents of the file stdio.h." You will almost always include one or more include files in your C programs. Lesson 22, "Advanced Compiler Use" presents more information about include files.

A variable definition informs the compiler of the variable's name and the type of information the variable is to hold.

The #define directive instructs the C compiler to replace a specific term with its assigned value throughout your program. By setting a variable at the top of your program and then using the term throughout the code, you can more easily change a term if needed by changing the single #define line as opposed to every place throughout the code. For example, if you wrote a payroll program that used a specific deduction for health insurance and the insurance rate changed, tweaking a variable created with #define named HEALTH_INSURANCE at the top of your program (or in a header file) would be so much easier than searching through lines and lines of code looking for every instance that had the information. Lesson 3, "Storing Information: Variables and Constants" covers the #define directive.

The Variable Definition (Line 5)

A variable is a name assigned to a location in memory used to store information. Your program uses variables to store various kinds of information during program execution. In C, a variable must be defined before it can be used. A variable definition informs the compiler of the variable's name and the type of information the variable is to hold. In the sample program, the definition on line 4, int year1, year2;, defines two variables—named year1 and year2—that each hold an integer value. Lesson 3 presents more information about variables and variable definitions.

The Function Prototype (Line 7)

A function prototype provides the C compiler with the name and arguments of the functions contained in the program. It appears before the function is used. A function prototype is distinct from a function definition, which contains the actual statements that make up the function. (Function definitions are discussed in more detail in "The Function Definition" section.)

Program Statements (Lines 12, 13, 14, 17, 19, 20, 22, and 28)

The real work of a C program is done by its statements. C statements display information onscreen, read keyboard input, perform mathematical operations, call functions, read disk files, and all the other operations that a program needs to perform. Most of this book is devoted to teaching you the various C statements. For now, remember that in your source code, C statements are generally written one per line and always end with a semicolon. The statements in bigyear.c are explained briefly in the following sections.

The printf() Statement

The printf() statement (lines 12, 13, 19, and 20) is a library function that displays information onscreen. The printf() statement can display a simple text message (as in lines 12 and 13) or a message mixed with the value of one or more program variables (as in lines 19-20).

The scanf() Statement

The scanf() statement (line 14) is another library function. It reads data from the keyboard and assigns that data to one or more program variables.

The program statement on line 17 calls the function named calcYear(). In other words, it executes the program statements contained in the function calcYear(). It also sends the argument year1 to the function. After the statements in calcYear() are completed, calcYear() returns a value to the program. This value is stored in the variable named year2.

The return Statement

Lines 22 and 28 contain return statements. The return statement on line 28 is part of the function calcYear(). It calculates the year a person would be a specific age by adding the #define constant TARGET_AGE to the variable year1 and returns the result to the program that called calcYear(). The return statement on line 22 returns a value of 0 to the operating system just before the program ends.

The Function Definition (Lines 26 Through 29)

When defining functions before presenting the program bigyear.c, two types of functions—library functions and user-defined functions—were mentioned. The printf() and scanf() statements are examples of the first category, and the function named calcYear(), on lines 26 through 29, is a user-defined function. As the name implies, user-defined functions are written by the programmer during program development. This function adds the value of a created constant to a year and returns the answer (a different year) to the program that called it. In Lesson 5, "Packaging Code in Functions," you learn that the proper use of functions is an important part of good C programming practice.

Note that in a real C program, you probably wouldn't use a function for a task as simple as adding two numbers. It has been done here for demonstration purposes only.

Program Comments (Lines 1, 11, 16, and 25)

Any part of your program that starts with /* and ends with */ or any single line that begins with // is called a comment. The compiler ignores all comments, so they have absolutely no effect on how a program works. You can put anything you want into a comment, and it won't modify the way your program operates. The first type of comment can span part of a line, an entire line, or multiple lines. Here are three examples:

```
/* A single-line comment */

int a,b,c; /* A partial-line comment */

/* a comment
spanning
multiple lines */
```

You should not use nested comments. A nested comment is a comment that has been put into another comment. Most compilers will not accept the following:

```
/*
/* Nested comment */
*/
```

Some compilers do allow nested comments. Although this feature might be tempting to use, you should avoid doing so. Because one of the benefits of C is portability, using a feature such as nested comments might limit the portability of your code. Nested comments also might lead to hard-to-find problems.

The second style of comment, the ones beginning with two consecutive forward slashes (//), are only for single-line comments. The two forward slashes tell the compiler to ignore everything that follows to the end of the line.

```
// This entire line is a comment
int x; // Comment starts with slashes
```

Many beginning programmers view program comments as unnecessary and a waste of time. This is a mistake! The operation of your program might be quite clear when you write the code; however, as your programs become larger and more complex, or when you need to modify a program you wrote 6 months ago, comments are invaluable. Now is the time to develop the habit of using comments liberally to document all your programming structures and operations. You can use either style of comments you prefer. Both are used throughout the programs in the book.

Do

DO add abundant comments to your program's source code, especially near statements or functions that could be unclear to you or to someone who might have to modify it later.

DO learn to develop a style that will be helpful. A style that's too lean or cryptic doesn't help. A style that is verbose may cause you to spend more time commenting than programming.

Don't

DON'T add unnecessary comments to statements that are already clear. For example, entering

```
/* The following prints Hello
World! on the screen */
printf("Hello World!);
```

might be going a little too far, at least when you're completely comfortable with the printf() function and how it works.

Using Braces (Lines 10, 23, 27, and 29)

You use braces {} to enclose the program lines that make up every C function—including the main() function. A group of one or more statements enclosed within braces is called a block. As you see in later lessons, C has many uses for blocks.

Running the Program

Take the time to enter, compile, and run bigyear.c. It provides additional practice in using your editor and compiler. Recall these steps from Lesson 1, "Getting Started with C":

Make your programming directory current.

Start your editor.

Enter the source code for bigyear.c exactly as shown in Listing 2.1, but be sure to omit the line numbers and colons.

Save the program file.

Compile and link the program by entering the appropriate command(s) for your compiler. If no error messages display, you can run the program by clicking the appropriate button in your C environment.

If any error messages display, return to step 2 and correct the errors.

A Note on Accuracy

A computer is fast and accurate, but it also is completely literal. It doesn't know enough to correct your simplest mistake; it takes everything you enter exactly as you entered it, not as you meant it!

This goes for your C source code as well. A simple typographical error in your program can cause the C compiler to choke, gag, and collapse. Fortunately, although the compiler isn't smart enough to correct your errors (and you'll make errors—everyone does!), it is smart enough to recognize them as errors and report them to you. (You saw in Lesson 1 how the compiler reports error messages and how you interpret them.)

A Review of the Parts of a Program

Now that all the parts of a program have been described, you can look at any program and find some similarities. Look at Listing 2.2 and see whether you can identify the different parts.

Input down-arrow.jpg

Listing 2.2 list_it.c – A Program to List a Code Listing with Added Line Numbers

```
 1:  /* list_it.c_ _This program displays a listing with line numbers! */
 2:  #include <stdio.h>
 3:  #include <stdlib.h>
 4:  #define BUFF_SIZE 256
 5:  void display_usage(void);
 6:  int line;
 7:
 8:  int main( int argc, char *argv[] )
 9:  {
10:    char buffer[BUFF_SIZE];
11:    FILE *fp;
12:
13:    if( argc < 2 )
14:    {
15:      display_usage();
16:      return (1);
```

Please Consider Serving on the NaSPA Board

Since 1986 the **N**etwork **a**nd **S**ystems **P**rofessionals **A**ssociation has provided Information Technology and Networking professionals worldwide with education, member discounts, job placement and award winning publications. We are looking for a few accomplished, experienced, and capable men and women willing to serve on our **Board of Directors.**

The mission of NaSPA, Inc., a not-for-profit organization, is to enhance the status and promote the advancement of all network and systems professionals; nurture members' technical and managerial knowledge and skills; improve members' professional careers through the sharing of technical information; promote the profession as a whole; further the understanding of the profession and foster understanding and respect for individuals within it; develop and improve educational standards; and assist in the continuing development of ethical standards for practitioners in the industry.

If you think you have what it takes to mentor the next generation of Information Technology professionals, we would like to hear from you. Please contact me directly at president@naspa.com and let’s begin the journey.

NaSPA®

```
17:   }
18:
19:   if (( fp = fopen( argv[1], "r" )) == NULL )
20:   {
21:         fprintf( stderr, "Error opening file, %s!", argv[1] );
22:       return(1);
23:   }
24:
25:   line = (1);
26:
27:  while( fgets( buffer, BUFF_SIZE, fp ) != NULL )
28:     fprintf( stdout, "%4d:\t%s", line++, buffer );
29:
30:   fclose(fp);
31:   return 0;
32: }
33:
34: void display_usage(void)
35: {
36:     fprintf(stderr, "\nProper Usage is: " );
37:     fprintf(stderr, "\n\nlist_it filename.ext\n" );
38: }
```

Output down-arrow.jpg

```
C:\>list_it list_it.c
 1:  /* list_it.c - This program displays a listing with line numbers! */
 2:  #include <stdio.h>
 3:  #include <stdlib.h>
 4:  #define BUFF_SIZE 256
 5:  void display_usage(void);
 6:  int line;
 7:
 8:  int main( int argc, char *argv[] )
 9:  {
10:      char buffer[BUFF_SIZE];
11:      FILE *fp;
12:
13:      if( argc < 2 )
14:      {
15:          display_usage();
16:          return (1);
17:      }
18:
19:      if (( fp = fopen( argv[1], "r" )) == NULL )
20:      {
21:            fprintf( stderr, "Error opening file, %s!", argv[1] );
22:          return(1);
23:      }
24:
25:      line = 1;
26:
27:         while( fgets( buffer, BUFF_SIZE, fp ) != NULL )
28:        fprintf( stdout, "%4d:\t%s", line++, buffer );
29:
30:      fclose(fp);
31:      return (0);
32:  }
33:
34:  void display_usage(void)
35:  {
36:      fprintf(stderr, "\nProper Usage is: " );
37:      fprintf(stderr, "\n\nlist_it filename.ext\n" );
38:  }
```

Analysis down-arrow.jpg

The list_it.c program in Listing 2.2 displays C program listings that you have saved. These listings display on the screen with line numbers added.

Looking at this listing, you can summarize where the different parts are. The required main() function is in lines 8 through 32. Lines 2 and 3 have #include directives. Lines 6, 10, and 11 have variable definitions. Line 4 defines a constant BUFF_SIZE as 256, the stand size for buffers. The value to doing this is that if the buffer size changes, you only need to adjust this one line and all lines using this constant will automatically update. If you hardcode a number like 256, you'd have to search all your lines of code to make sure you caught all mentions.

A function prototype, void display_usage(void), is in line 5. This program has many statements (lines 13, 15, 16, 19, 21, 22, 25, 27, 28, 30, 31, 36, and 37). A function definition for display_usage() fills lines 34 through 38. Braces enclose blocks throughout the program. Finally, only line 1 has a comment. In most programs, you should probably include more than one comment line.

list_it.c calls many functions. It calls only one user-defined function, display_usage(). The library functions that it uses are fopen() in line 19; fprintf() in lines 21, 28, 36, and 37; fgets() in line 27; and fclose() in line

30. These library functions are covered in more detail throughout this book.

Summary

This lesson was short, but it's important because it introduced you to the major components of a C program. You learned that the single required part of every C program is the main() function. You also learned that a program's real work is done by program statements that instruct the computer to perform your desired actions. You were also introduced to variables and variable definitions, and you learned how to use comments in your source code.

In addition to the main() function, a C program can use two types of subsidiary functions: library functions, supplied as part of the compiler package, and user-defined functions, created by the programmer. The next few lessons go into much more detail on many of the parts of a C program that you saw in this lesson.

Q&A

Q What effect do comments have on a program?

A Comments are for programmers. When the compiler converts the source code to object code, it throws the comments and the white space away. This means that they have no effect on the executable program. A program with a lot of comments executes just as fast as a program with few comments. Comments do make your source file bigger, but this is usually of little concern. To summarize, you should use comments and white space to make your source code as easy to understand and maintain as possible.

Q What is the difference between a statement and a block?

A A block is a group of statements enclosed in braces ({}). A block can be used in most places that a statement can be used.

Q How can I find out what library functions are available?

A Many compilers come with online documentation dedicated specifically to documenting the library functions. They are usually in alphabetical order. Appendix C, "Common C Functions," lists many of the available functions. After you begin to understand more of C, it would be a good idea to read that appendix so that you don't rewrite a library function. (There's no use reinventing the wheel!)

Workshop

The Workshop provides quiz questions to help you solidify your understanding of the material covered and exercises to provide you with experience in using what you've learned.

Quiz

What is the term for a group of one or more C statements enclosed in braces?

What is the one component that must be present in every C program?

How do you add program comments, and why are they used?

What is a function?

C offers two types of functions. What are they, and how are they different?

What is the #include directive used for?

Can comments be nested?

Can comments be longer than one line?

What is another name for an include file?

What is an include file?

Exercises

Write the smallest program possible.

Consider the following program:

```
1:  /* ex02-02.c */
2:  #include <stdio.h>
3:
4:  void display_line(void);
5:
6:  int main(void)
7:  {
8:     display_line();
9:        printf("\n Teach Yourself C In One Hour a Day!\n");
10:    display_line();
11:
12:     return 0;
13: }
14:
15: /* print asterisk line */
16: void display_line(void)
17: {
18:    int counter;
19:
20:    for( counter = 0; counter < 30; counter++ )
21:        printf("*" );
22: }
23: /* end of program */
```

What line(s) contain statements?

What line(s) contain variable definitions?
What line(s) contain function prototypes?
What line(s) contain function definitions?
What line(s) contain comments?
Write an example of a comment.
What does the following program do? (Enter, compile, and run it.)

```
1:  /* ex02-04.c */
2:  #include <stdio.h>
3:
4:  int main(void)
5:  {
6:     int ctr;
7:
8:     for( ctr = 65; ctr < 91; ctr++ )
9:         printf("%c", ctr );
10:
11:    printf("\n");
11:    return 0;
12: }
13: /* end of program */
```

What does the following program do? (Enter, compile, and run it.)

```
1:  /* ex02-05.c */
2:  #include <stdio.h>
3:  #include <string.h>
4:  int main(void)
5:  {
6:     char buffer[256];
7:
8:     printf( "Enter your name and press <Enter>:\n");
9:     fgets( buffer );
10:
11:       printf( "\nYour name has %d characters and spaces!",
12                     strlen( buffer ));
13:
14:    return 0;
15: }
```

Bradley L. Jones is the site manager for a number of high profile developer sites including CodeGuru.com, Developer.com, and Javascripts.com and he is an Executive Editor of internet.com's EarthWeb channel. Bradley has been working with C# longer than most developers since he was invited to Microsoft prior to the official beta release. Bradley's background includes experience developing in C, C++, PowerBuilder, SQL Server, and numerous other tools and technologies. Additionally, he is an internationally best selling author who wrote the original 21 Days book -- Sams Teach Yourself C in 21 Days.

Peter Aitken has been writing about computers and programming for more than 10 years, with some 30 books and more than 1.5 million copies in print as well as hundreds of magazine and trade publication articles. His book titles include Sams Teach Yourself Internet Programming With Visual Basic in 21 Days and Sams Teach Yourself C in 21 Days. A regular contributor to Office Pro magazine and the DevX Web site, Peter is the proprietor of PGA Consulting, providing custom application and Internet development to business, academia, and government since 1994. You can reach him at peter@pgacon.com.

Dean Miller is a writer and editor with more than 20 years of experience in both the publishing and licensed consumer product businesses. Over the years, he has created or helped shape a number of bestselling books and series, including Teach Yourself in 21 Days, Teach Yourself in 24 Hours, and the Unleashed series, all from Sams Publishing. He has written books on C programming and professional wrestling, and is still looking for a way to combine the two into one strange amalgam.

"You May Be Entitled to a Cash Recovery"

Things Technology Companies Should Know About Asset Protection, Dispute Resolution, and Disaster Recovery

Damage Claim, Financial Dispute, or Disaster?

The "right" Experts can improve chances for financial recovery from data center disasters, cable cuts, billing disputes, and other complex technology claims.

The **Leo A. Wrobel Companies** are your entry point to a nationwide network of Experts. We are not a law firm. We are the technical Experts who do the heavy lifting in complex disputes, loss claims and lawsuits. In many cases you pay nothing unless you collect.

If your organization has sustained a loss and seeks financial recovery, call us *first* before you call a law firm. We have recovered millions in damage, performance, contract, and billing dispute claims since 1999 - *often without any litigation*.

In cases where litigation is unavoidable, we work with law firms that *win cases* because they employ the "right" Experts ... *Like Us*.

So why not get started right now? Call **1(214) 888-1300** for a confidential assessment of your claim, dispute, or disaster. After all,

"Found Money is a Good Thing." ™

Tel*LAW*Com Labs Inc. specializes in financial dispute resolution with experience in claims from $30,000 to $200 million. www.tlc-labs.com

***b4Ci* Inc.** helps manage risk by writing disaster recovery plans, conducting business impact analysis and on site training. www.b4Ci.com

ROW911 provides financial recovery for fiber and telephone cable cuts, as well as pipeline, electric and other damage claims, for owners and affected end users. www.row911.com

Honesty Integrity Knowledge Experience Credibility Reputation