Advanced Software Techniques

Regular Expressions

What's a Regular Expression?

A powerful and flexible way of matching patterns of text in another string

We saw why we might want to do this in the Input Validation assignment

Doing it manually
is a pain!

Regular expressions
can make it
easier

History

grep (1973)

vi (1976)

awk (1977)

Perl (1987)

Basic Concept

1. Make a pattern
2. Match the pattern to a substring

Example from grep

grep "^ABC.*5$"

means

"search for all strings

that start with ABC, then have 0 or more of any characters, and then end in 5"

Example from vi

1,$s/^.* /nick& &/

means

"find any string that starts with any character

and has a space in it

and duplicate that part of the string, preceding it by nick"

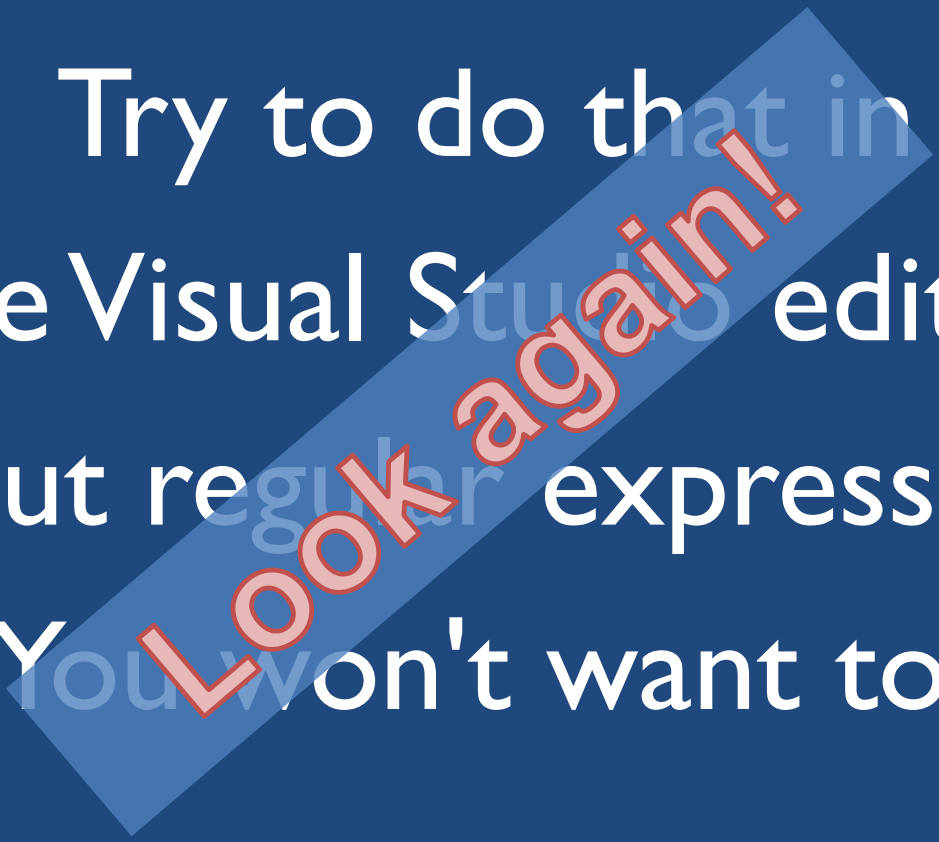
Why?

To easily create sample nicknames (DSAD #2)

Regular Expressions are Powerful!

Try to do that in the Visual Studio editor without regular expressions ...

You won't want to!



How about using them in a program?

There are many competing libraries

- cppre
- DEELX
- GLib
- GRETA
- ICU
- Oniguruma
- PCRE
- OT
- regex
- re2
- TRE

And TRI

We'll be looking at TRI

TR1

"Technical Report 1: proposed extensions to the C++ Standard Library"

Built in to
Visual C++ 2010

Yay!

And It Just Works!

#include <regex>

Declare a regex object:

```
regex reg("substr");
```

Search using regex_search():

regex_search(str.begin(), str.end(), reg);

str is a string; reg is a regex object

Example from cpp0x.googlecode.com/svn-history/r2/trunk/reg.cpp

```
#include<regex>
#include<string>
#include<iostream>
#include<cassert>
using namespace std;
int main()
{
```

```
cout << "Program is started!" << endl;
std::string str = "Hello world";
regex rx("ello");
assert( regex_search(str.begin(), str.end(), rx) );
}
```

This will tell you if

"ello"

is in

"Hello world"

OK, so?

Why not use strstr() on a C-style null-terminated string or find() on a C++ string?

```
std::string str = "Hello world";
regex rx("e.*o");
assert( regex_search(str.begin(), str.end(), rx) );
```

Is there a substring starting with an e and ending in an o?

OK!

There are 6 different regular expression syntaxes in TR1

- ECMAScript
- basic
- extended
- awk
- grep
- egrep

We'll do some
of the highlights
(not enough time!)

Advance Credit:

Many of the concepts are from a PPT found at

http://snap.nlc.dcccd.edu/learn/frazer1

(Web site no longer available.)

Another decent reference:

http://msdn.microsoft.com/en-us/library/bb982727.aspx

NOTE #1: Always:

```
#include <regex>
using namespace std;
```

NOTE #2: If you ever need to use a literal version of any of the following, precede it by \

Wildcard Characters

. means any character

Character Classes

[ ] can be used to
specify a set of characters
e.g. [aeiou] specifies a vowel

It behaves like the
scanf() family,
supporting ^ for inversion
and - for ranges

Some syntaxes
support already named classes

e.g.:

[[:alpha:]]

[[:alnum:]]

[[:digit:]]

[[:lower:]]

And [[:alnum:]] is also the same as [[:w:]]

(representing a word)

And some of them have further shortforms too:

e.g.:

[[:digit:]] is shortened to \d

[[:word:]] is shortened to \w

Anchoring

^ indicates the start of the string

$ indicates the end of the string

Optionality

? following an expression means that the previous expression might or might not be there

e.g. “honou?r”

Repetition

* following an expression means 0 or more occurrences of that expression

Pattern	Meaning
.*	Any character, 0 or more times
,*	0 or more commas
y*	0 or more y characters

+ means one or more occurrences

Repetition Ranges

Instead of using * for 0 or more occurrences, you can use {} to specify a range of occurrences

Pattern	Meaning
a{5}	5 occurrences of the letter a
a{1,5}	at least 1 or as many as 5 occurrences of the letter a
a{4,}	at least 4 occurrences of the letter a
[[:alpha:]]{4,}	at least 4 alphabetic characters

Subexpressions

And you can group expressions using ( )

e.g.

(abcd*){3} means

abc and any number of d characters

exactly 3 times

Alternation

The | character acts like in a C condition, allowing for one item OR the other item

e.g. (yes|no|maybe)

Some TR1 Methods

regex_match() does
a match of
an entire string
against a
regular expression

regex_search() does
a match of
parts of a string
against a
regular expression

Both
regex_match()
and
regex_search()
return true/false as a bool

The parameters vary, depending on which overloaded version you use

A common version takes:

- the main string
- results (passed as a reference and filled in)
- the string to look for

The results reference is of type cmatch

and contains

an array of items found

Examples

Source: from

http://www.codeguru.com/cpp/cpp/cpp_mfc/stl/article.php/c15339/A-TRI-Tutorial-Regular-Expressions.htm

Example #1: Validating an e-mail address (sort of)

```
bool is_email_valid(const std::string& email)
{
    // define a regular expression
    const std::tr1::regex pattern("(\\w+)(\\.|_)?(\\w*)@(\\w+)(\\.(\\w+))+");
    // try to match the string with the regular expression
    return std::tr1::regex_match(email, pattern);
}
```

Example #2: Extracting IP address parts

```
void show_ip_parts(const std::string& ip)
{
    // regular expression with 4 capture groups defined with
    // parenthesis (...)
    const std::tr1::regex pattern("(\\d{1,3}):(\\d{1,3}):(\\d{1,3}): (\\d{1,3})");
    // object that will contain the sequence of sub-matches
    std::tr1::match_results<std::string::const_iterator> result;
    // match the IP address with the regular expression
    bool valid = std::tr1::regex_match(ip, result, pattern);
    std::cout << ip << " \t: " << (valid ? "valid" : "invalid") << std::endl;
```

```
    // if the IP address matched the regex, then print the parts
    if(valid)
    {
        std::cout << "b1: " << result[1] << std::endl;
        std::cout << "b2: " << result[2] << std::endl;
        std::cout << "b3: " << result[3] << std::endl;
        std::cout << "b4: " << result[4] << std::endl;
    }
}
```

Example #3: Using regex_search() to find the first occurrence of a target

```
int main()
{
    const std::tr1::regex pattern("(\\w+day)");
    std::string weekend = "Saturday and Sunday";
    std::tr1::smatch result;

    bool match = std::tr1::regex_search(weekend, result, pattern);
```

```
    if(match)
    {
        // if there was a match print it
        for(size_t i = 1; i < result.size(); ++i)
        {
            std::cout << result[i] << std::endl;
        }
    }

    return 0;
}
```

Conclusion

As you can probably guess, there's a lot more than this

There's a lot of information out there

You just have to look