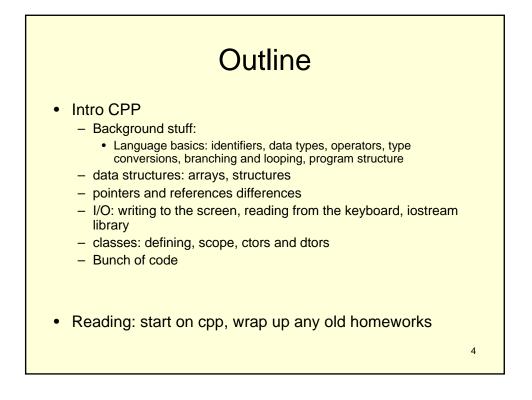


Reading

- Chapter 15,16 c++, data abstraction
- Chapter 17,18 classes





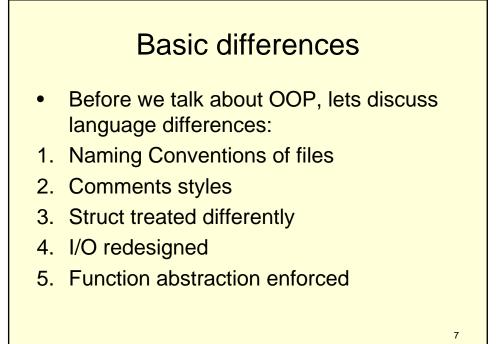
differences between c++ and c

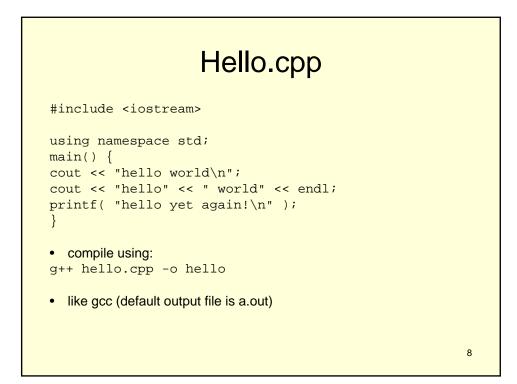
- history and background
- object-oriented programming with classes
- very brief history...
 - C was developed 69-73 at Bell labs.
 - C++ designed by Bjarne Stroustrop at AT&T Bell Labs in the early 1980's

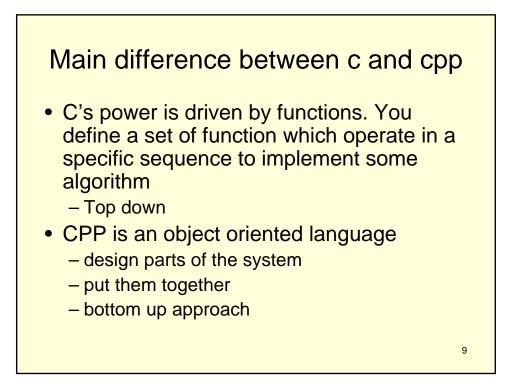
5

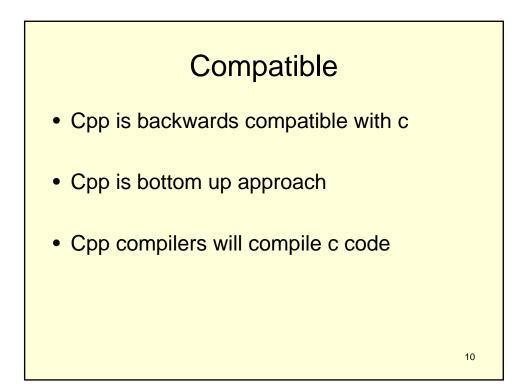
- originally developed as "C with classes"
- Idea was to create reusable code
- development period: 1985-1991
- ANSI standard C++ released in 1991

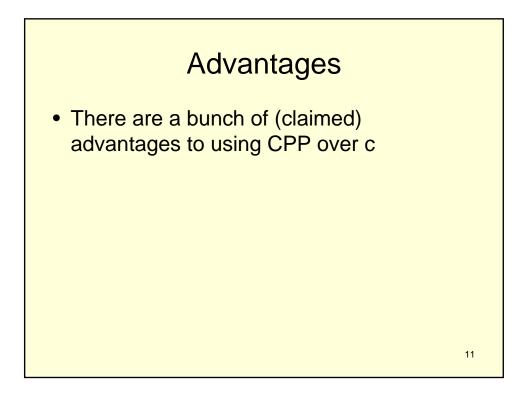
Four main OOP concepts abstraction - creation of well-defined interface for an object, separate from its implementation - e.g., Vector in Java e.g., key functionalities (init, add, delete, count, print) which can be called independently of knowing how an object is implemented encapsulation - keeping implementation details "private", i.e., inside the implementation hierarchy - an object is defined in terms of other objects - Composition => larger objects out of smaller ones Inheritance => properties of smaller objects are "inherited" by larger objects polymorphism - use code "transparently" for all types of same class of object - i.e., "morph" one object into another object within same hierarchy 6

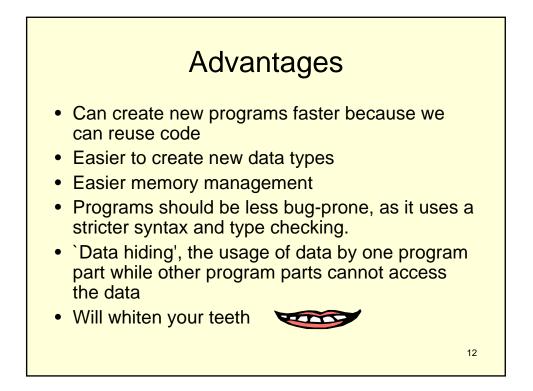


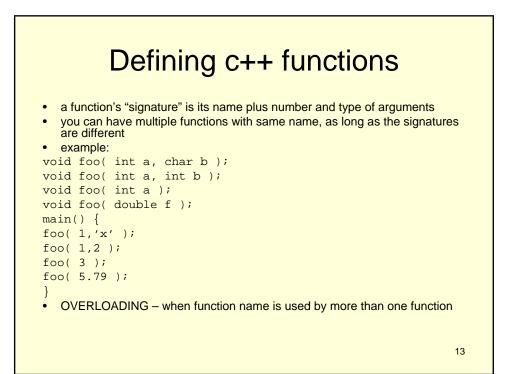


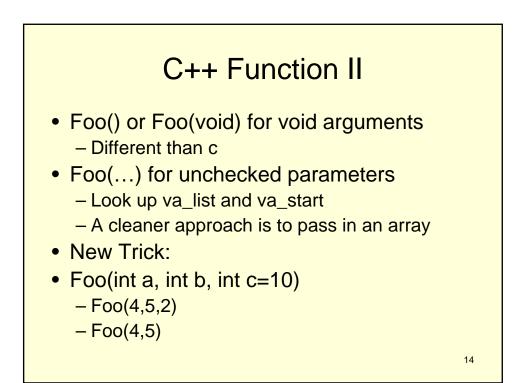


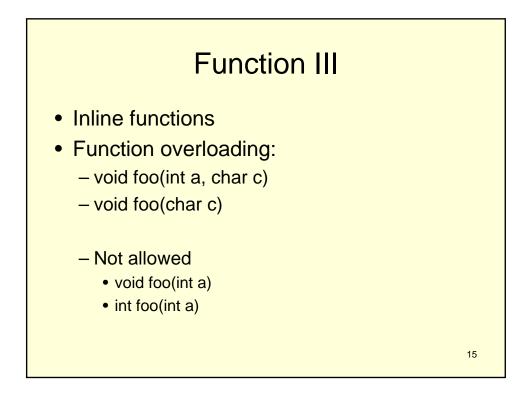


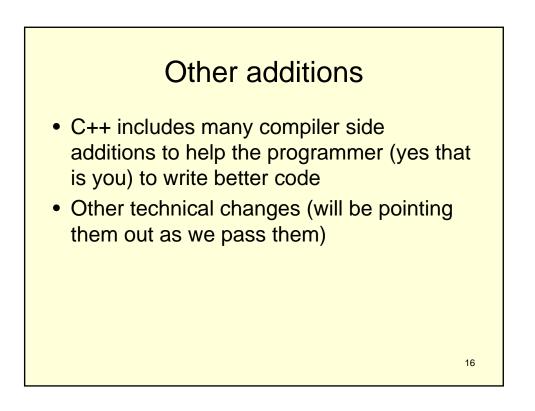


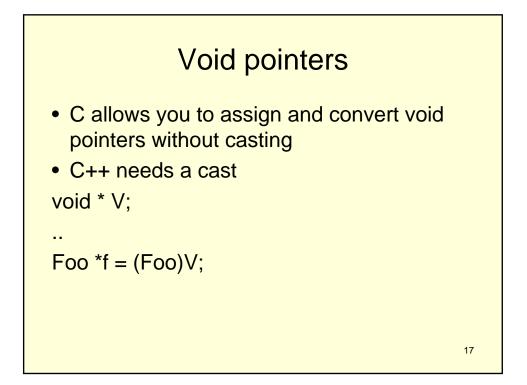


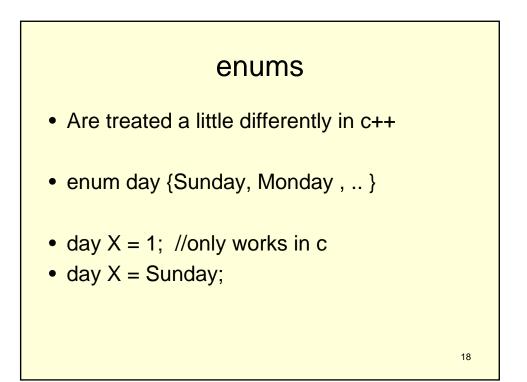


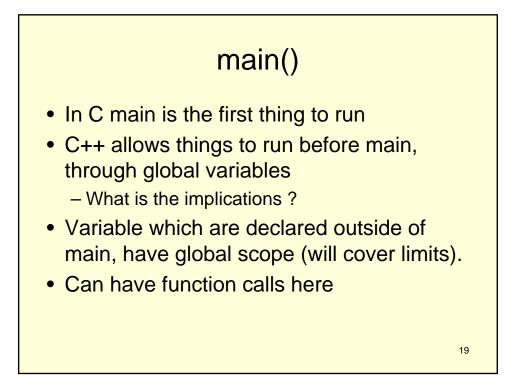


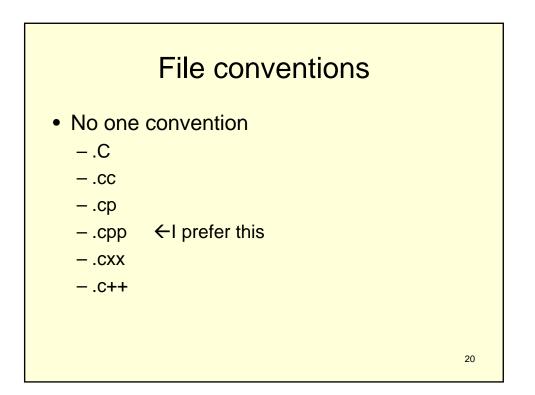


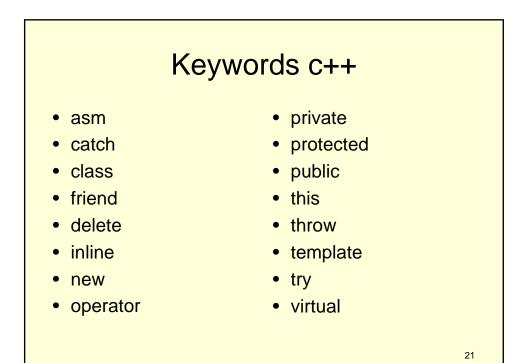


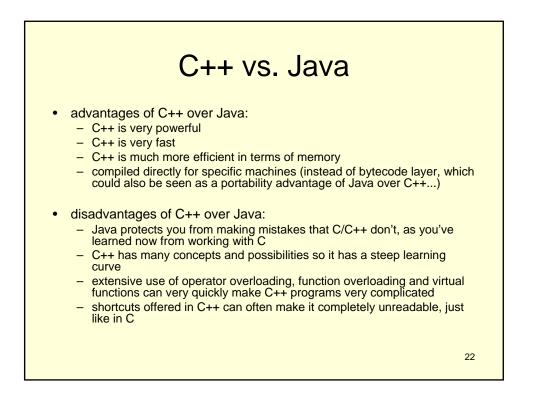


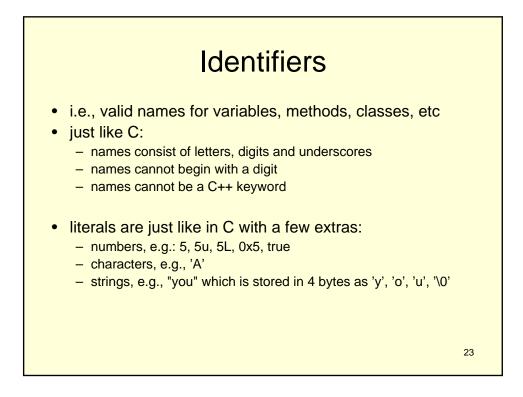


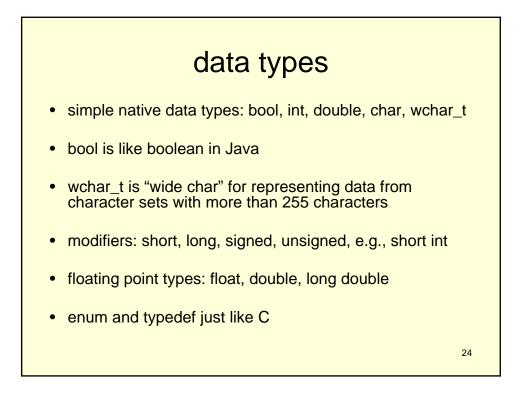


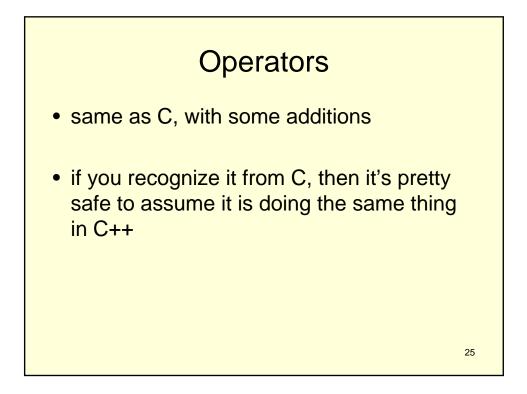


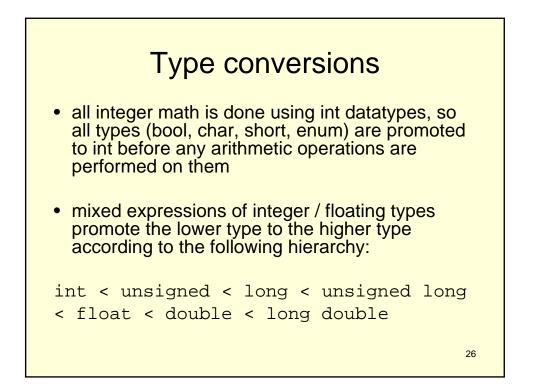


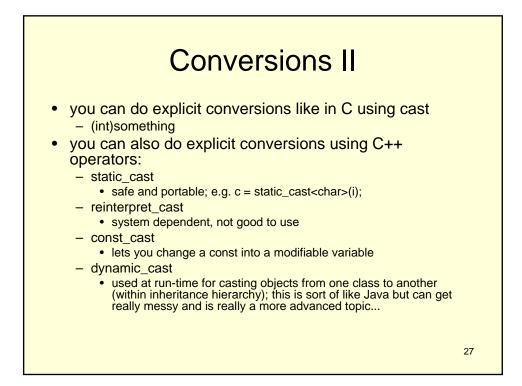


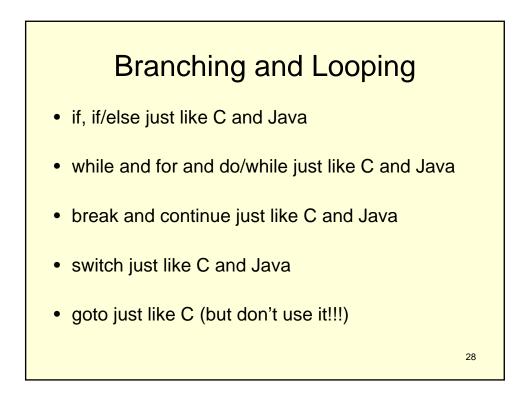


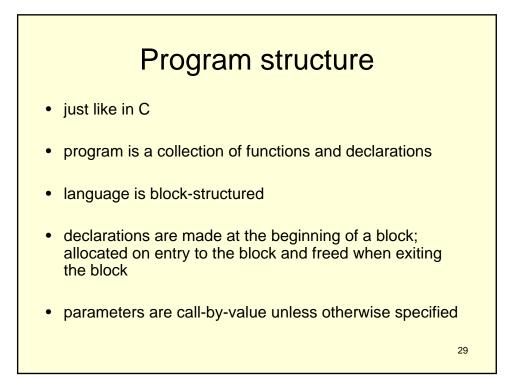


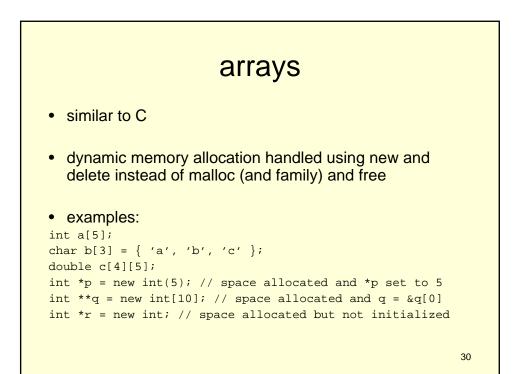


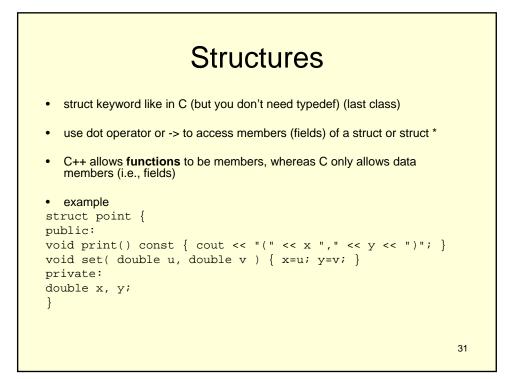


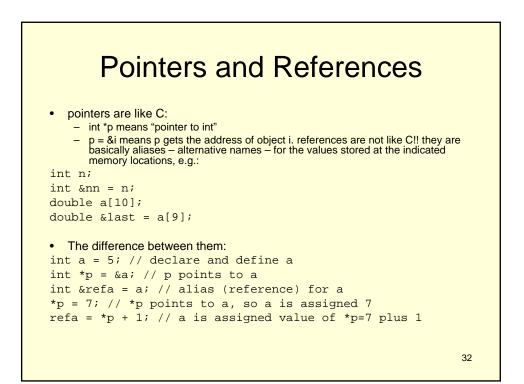


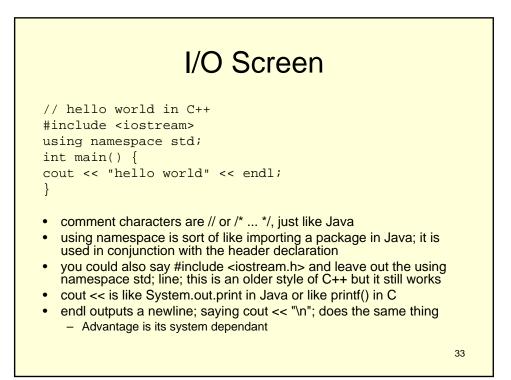


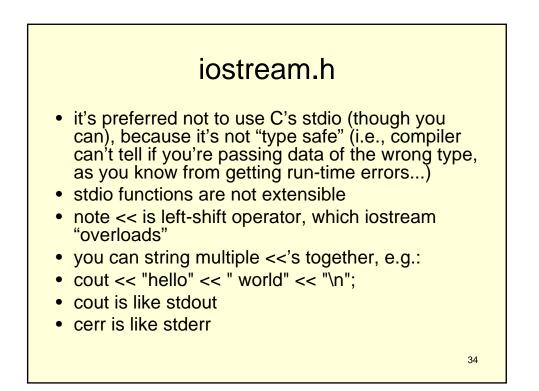










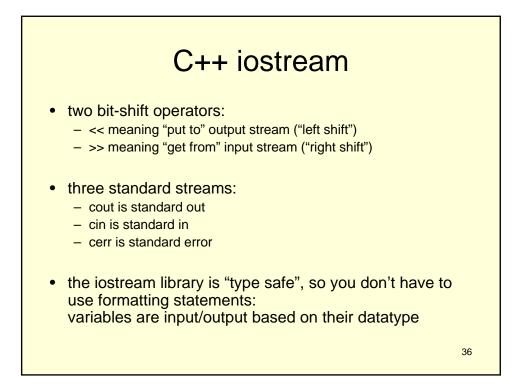


I/O keyboard read from the keyboard using cin >>, which is like scanf() • example:

```
#include <iostream>
using namespace std;
int main() {
int i;
cout << "enter a number: ";</pre>
cin >> i;
cout << "you entered " << i <<"\n";</pre>
}
```

•

in C



ostream and istream

• ostream

- cout is an ostream, << is an operator
- use cout.put(char c) to write a single char
- use cout.write(const char *p, int n) to write n chars
- use cout.flush() to flush the stream

istream

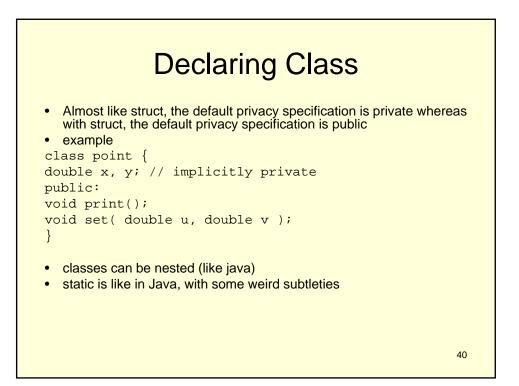
- cin is an istream, >> is an operator
- use cin.get(char &c) to read a single char
- use cin.get(char *s, int n, char c='\n') to read a line (inputs into string s at most n-1 characters, up to the specified delimiter c or an EOF; a terminating 0 is placed at the end of the input string s)

37

- also cin.getline(char *s, int n, char c='\n')
- use cin.read(char *s, int n) to read a string

<text><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item><list-item>

<section-header><section-header><section-header><section-header><text>



Using

```
point x;
x.set(3,4);
x.print();
```

point *pptr = &x;

```
pptr->set(3,2);
pptr->print();
```

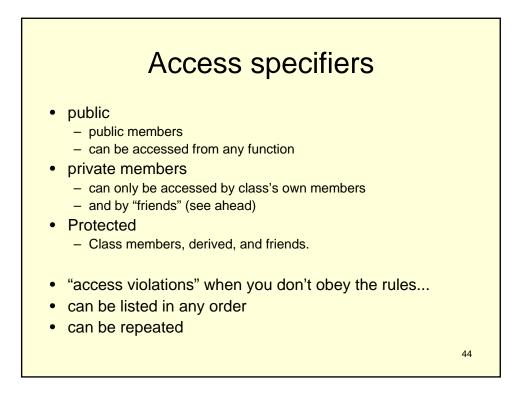
Classes: function overloading and overriding

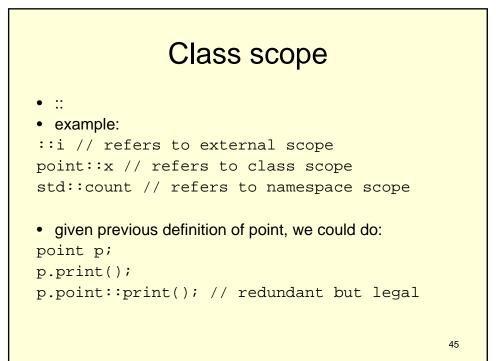
• overloading:

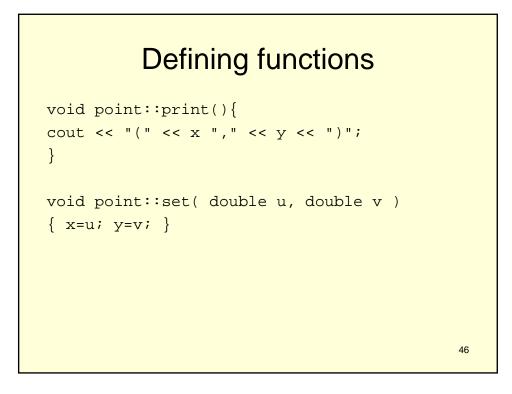
- when you use the same name for functions with different signatures
- functions in derived class supercede any functions in base class with the same name
- overriding:
 - when you change the behavior of base-class function in a derived class
 - DON'T OVERRIDE BASE-CLASS FUNCTIONS!!
- · because compiler can invoke wrong version by mistake
- but init() is okay to override
- (more explanation in ch 12...)

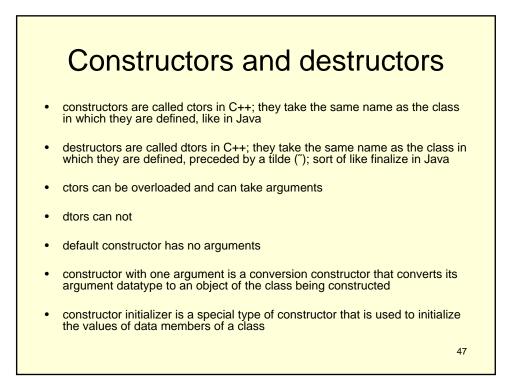
42



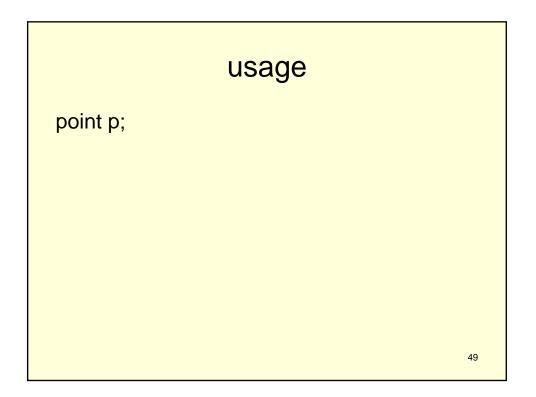


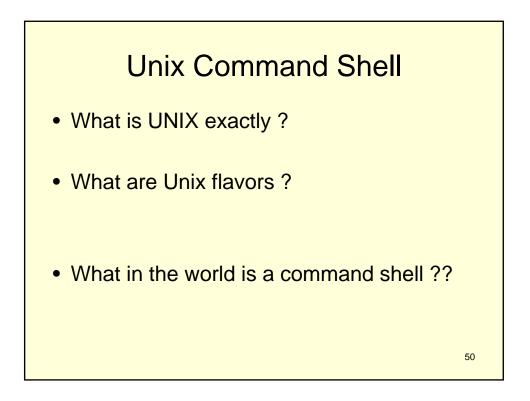


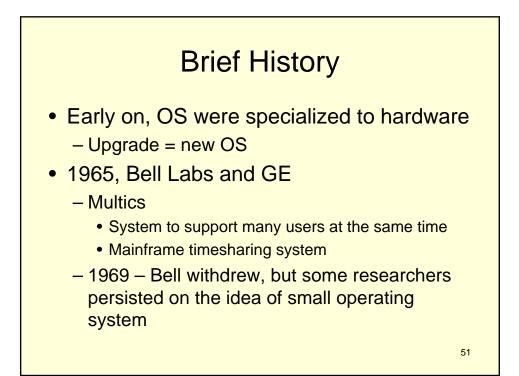


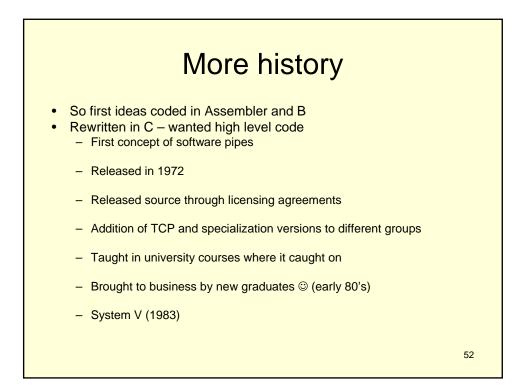


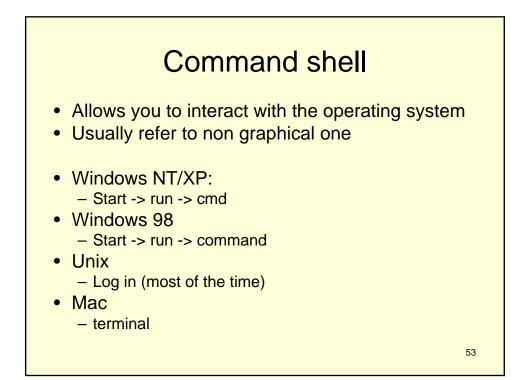
```
class point {
double x,y;
public:
point() { x=0;y=0; } // default
point( double u ) {x =u; y=0; }
// conversion
point( double u, double v )
        { x =u; y =v; }
.
.
.
.
.
.
```

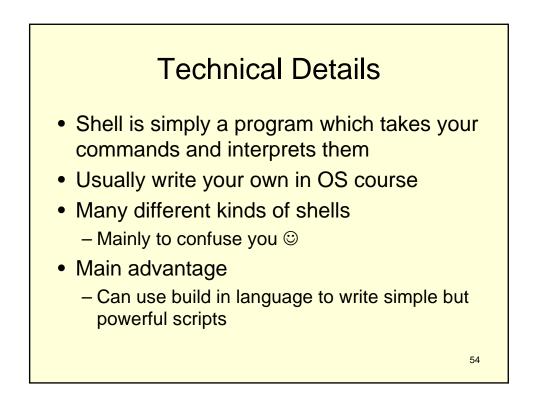


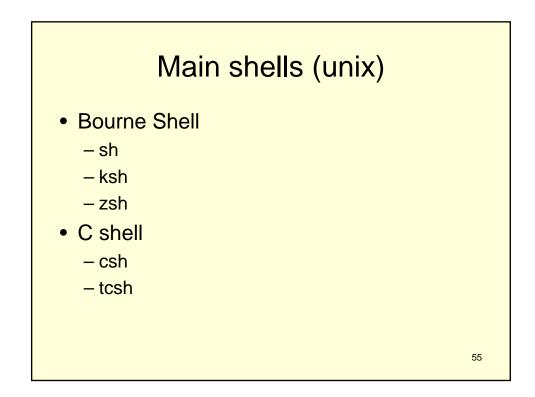


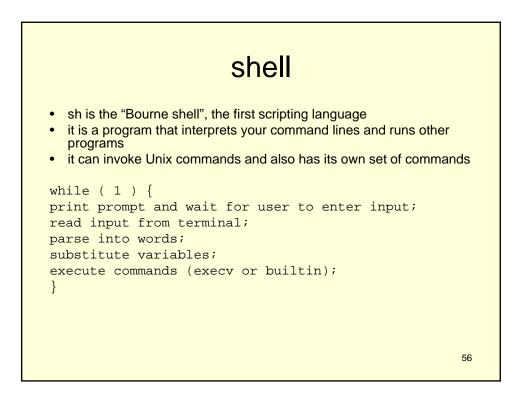


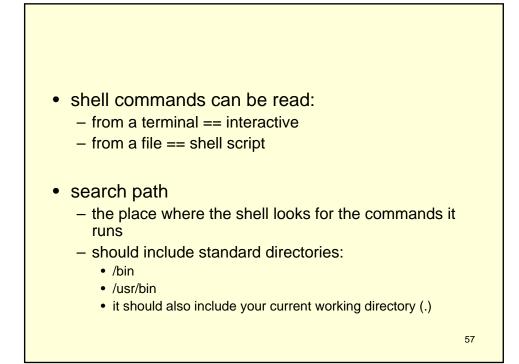


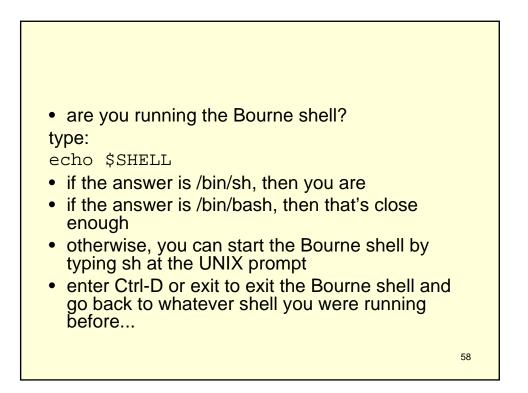


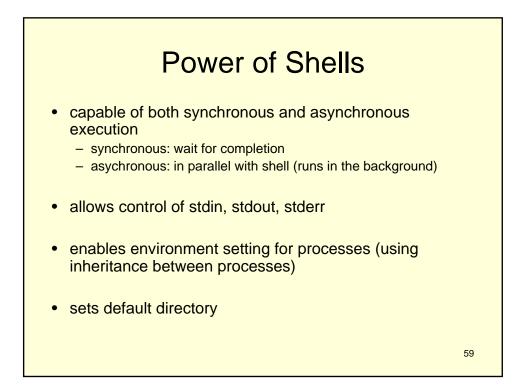


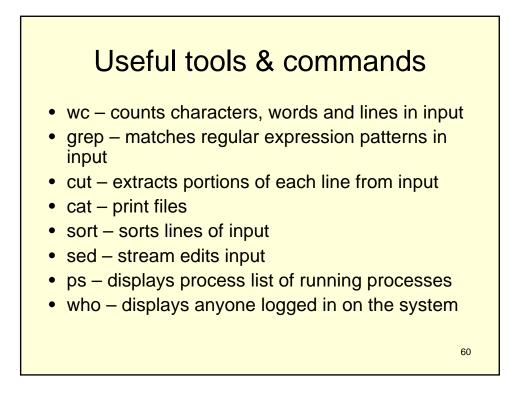


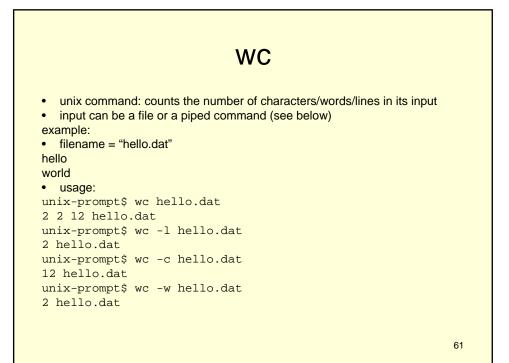






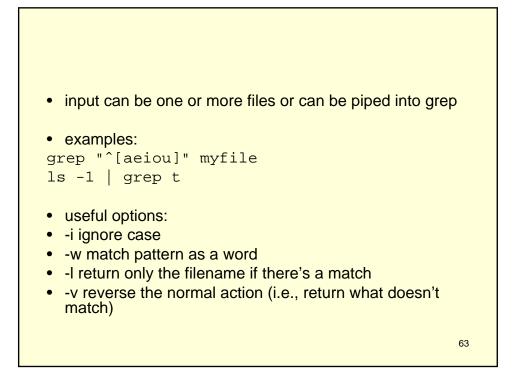






Global Regular Expression Parser GREP

- · one of the most useful tools in unix
- three standard versions:
 - plain old grep
 - extended grep: egrep
 - fast grep: fgrep
- used to search through files for ... regular expressions!
- prints only lines that match given pattern
- a kind of filter
- BUT it's line oriented

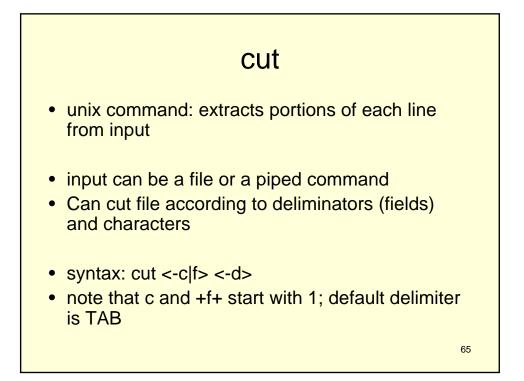


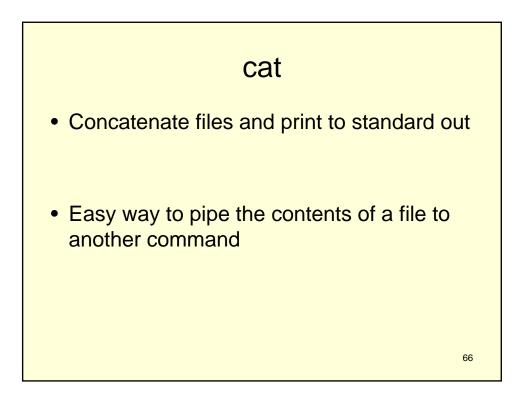
```
examples:

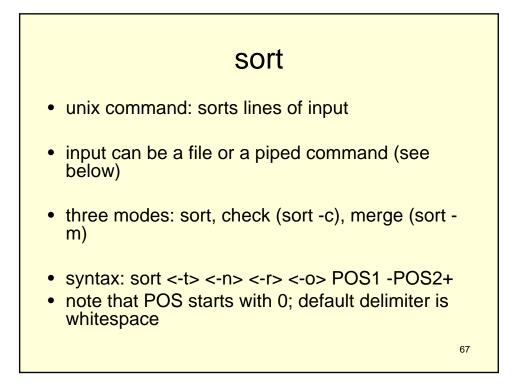
grep f f [aeiou] myfile

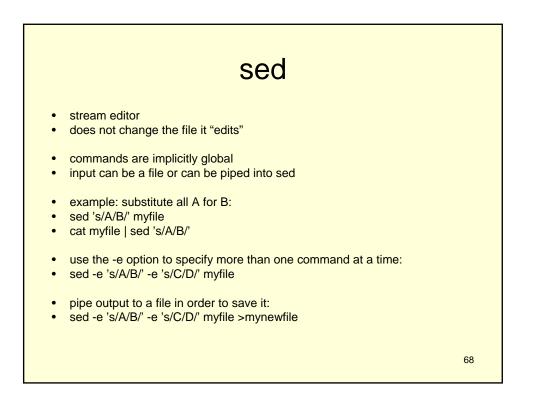
grep f f [aeiou] myfile

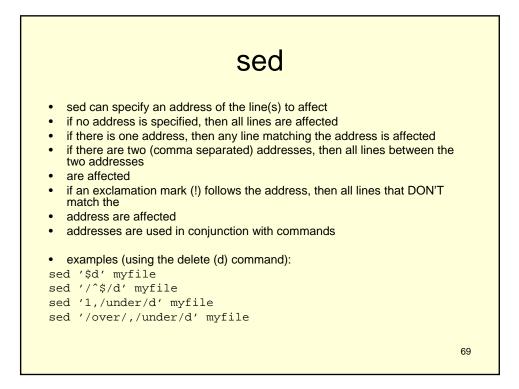
grep f f [aeiou] myfile
how do you list all lines containing a digit?
how do you list all lines containing a f
how do you list all lines containing a 0?
how do you list all lines containing a 5 and an 0?
```

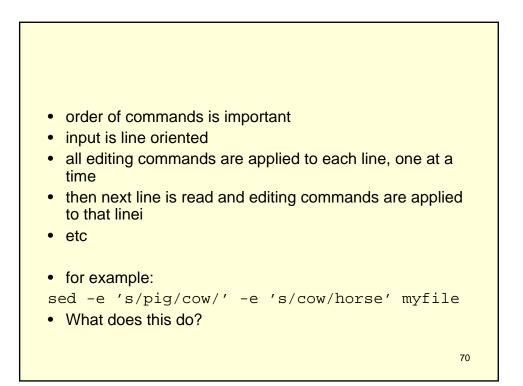


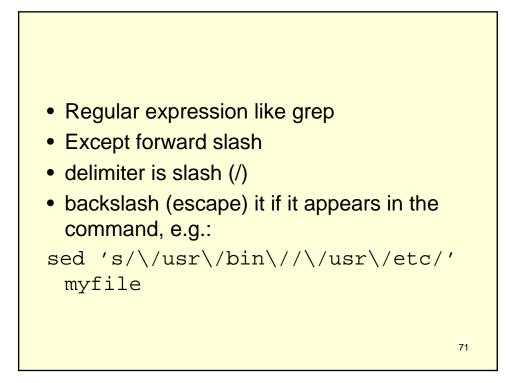


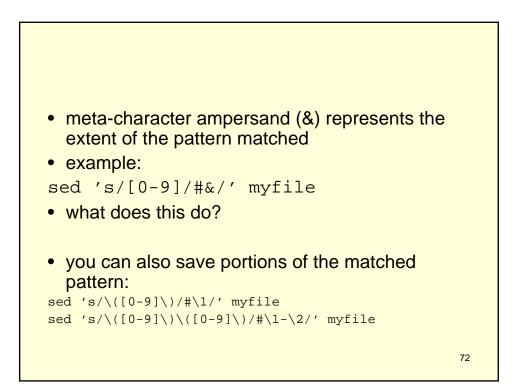


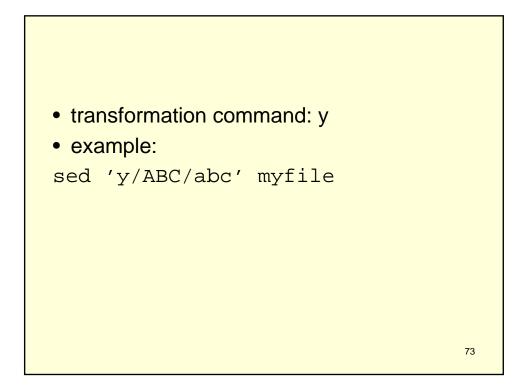


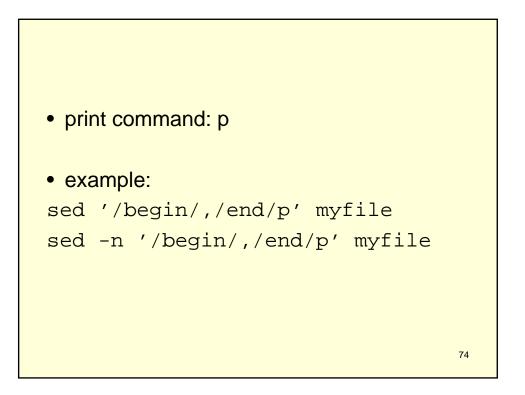


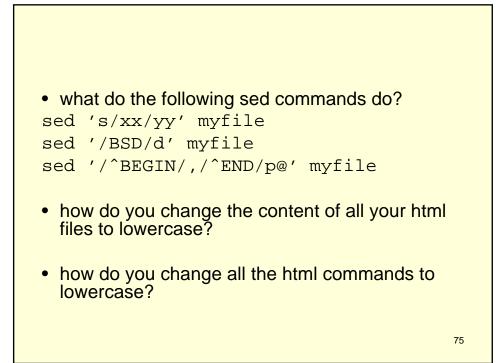


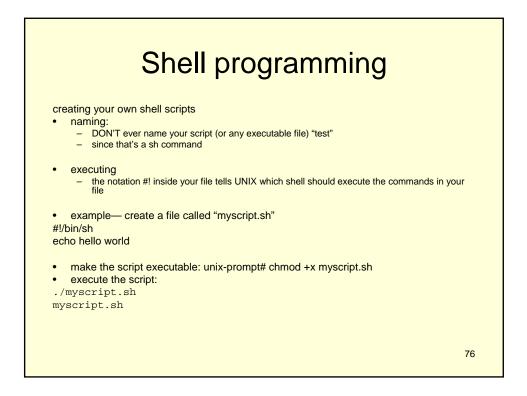


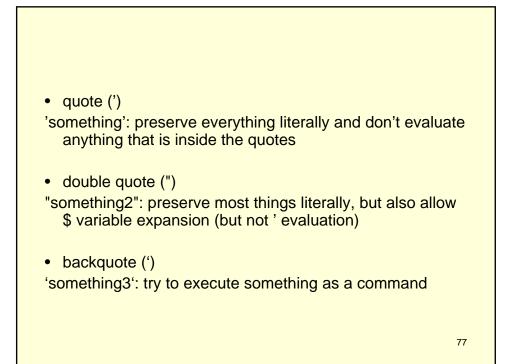












```
Filename is t.sh
• #!/bin/sh
• hello="hi"

    echo 0=$hello

 echo 1='$hello'

• echo 2="$hello"

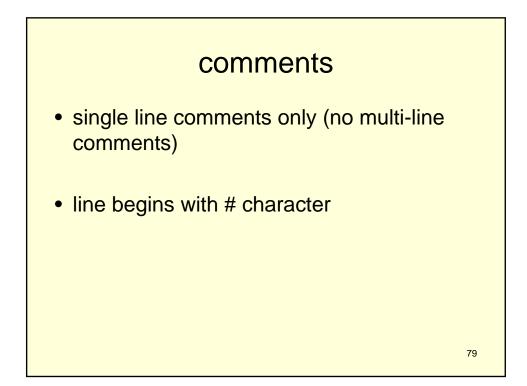
 echo 3=`$hello`

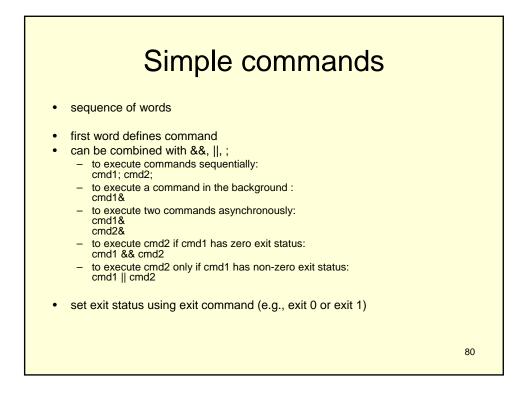
 echo 4="`$hello`"

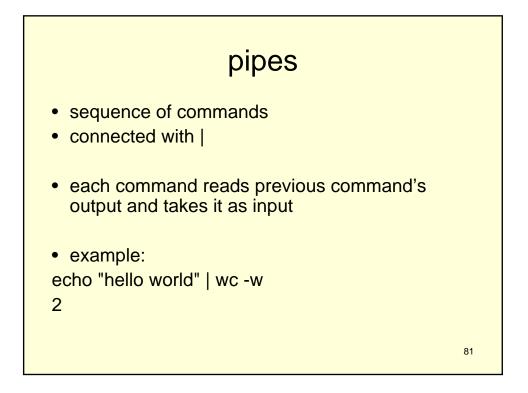
• echo 5="'$hello'"
• filename=hi
• #!/bin/sh

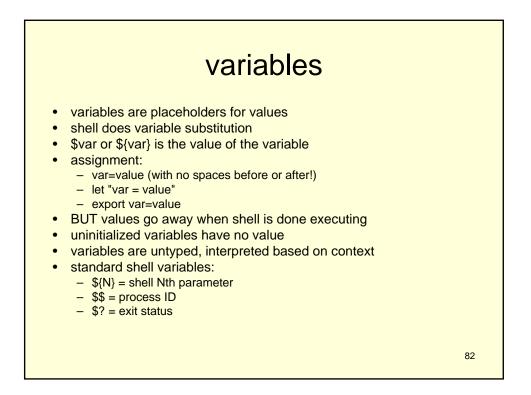
    echo "how did you get in here?"

output=
unix$ t.sh
0=hi
1=$hello
2=hi
3=how did you get in here?
4=how did you get in here?
5='hi'
                                                                       78
```



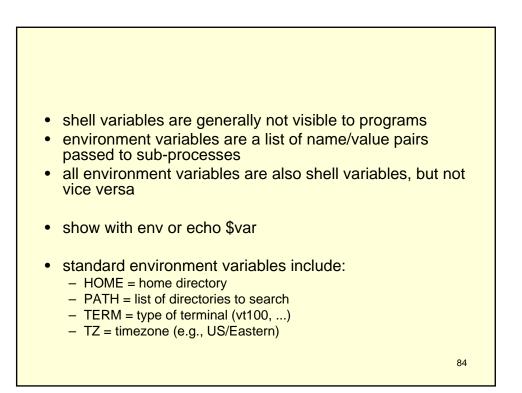


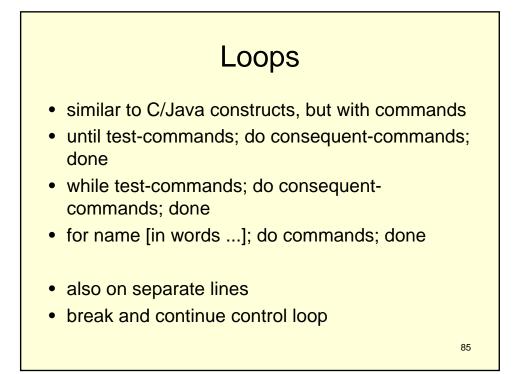




```
    filename=u.sh

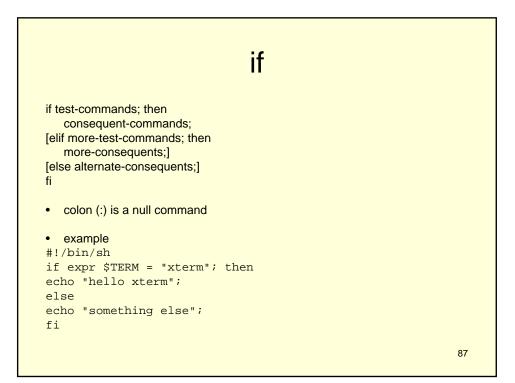
#!/bin/sh
echo 0=$0
echo 1=$1
echo 2=$2
echo 3=$$
echo 4=$?
• output
unix$ u.sh
0=.//u.sh
1=
2=
3=21093
4=0
unix$ u.sh abc 23
0=.//u.sh
1=abc
2=23
3=21094
4=0
                                                                      83
```

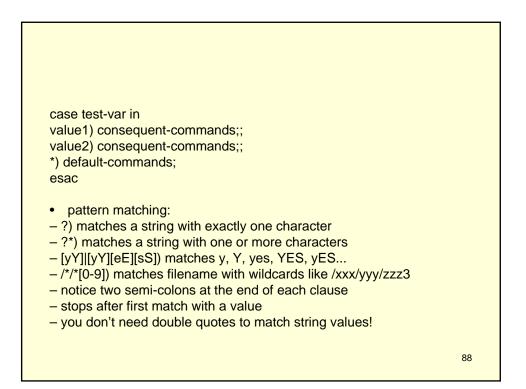




```
while
i=0
while [ $i -lt 10 ]; do
echo "i=$i"
((i=$i+1)) # same as let "i=$i+1"
done

for
for
for counter in `ls *.c`; do
echo $counter
done
```





example

```
#!/bin/sh
case "$TERM" in
xterm) echo "hello xterm";;
vt100) echo "hello vt100";;
*) echo "something else";;
esac
```

<section-header><section-header><list-item><list-item><list-item><list-item><list-item><list-item>
 biggest difference from traditional programming languages
 shell substitutes and executes
 order:

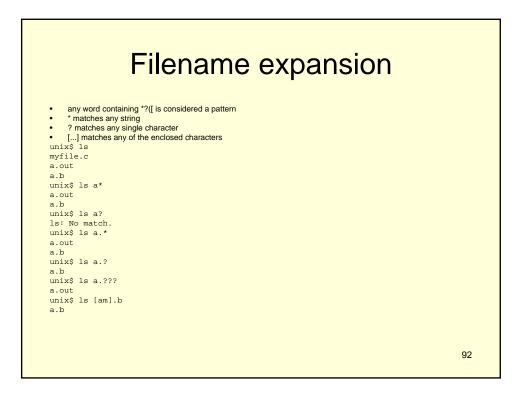
 brace expansion
 bide expansion
 command substitution
 arithmetic expansion
 word splitting

89

Command subing

replace \$(command) or 'command' by stdout of executing command • can be used to execute content of variables: unix\$ x=ls unix\$ \$x myfile.c a.out unix\$ echo \$x ls unix\$ echo `ls` myfile.c a.out unix\$ echo `x` sh: x: command not found unix\$ echo `\$x` myfile.c a.out unix\$ echo \$(ls) myfile.c a.out unix\$ echo \$(x) sh: x: command not found unix\$ echo \$(\$x) myfile.c a.out

91



redirection

- stdin, stdout and stderr may be redirected
- < redirects stdin (0) to come from a file
- > redirects stdout (1) to go to file
- >> appends stdout to the end of a file
- &> redirects stderr (2)
- >& redirects stdout and stderr, e.g.: 2>&1 sends stderr to the same place that stdout is going
- << gets input from a here document, i.e., the input is what you type, rather than reading from a file

93

Built in commands alias, unalias - create or remove a pseudonym or shorthand for a command or series of commands jobs, fg, bg, stop, notify - control process execution command — execute a simple command cd, chdir, pushd, popd, dirs - change working directory echo — display a line of text history, fc - process command history list set, unset, setenv, unsetenv, export - shell built-in functions to determine the characteristics for environmental variables of the current shell and its descendents getopts — parse utility options hash, rehash, unhash, hashstat — evaluate the internal hash table of the contents of directories kill — send a signal to a process 94

