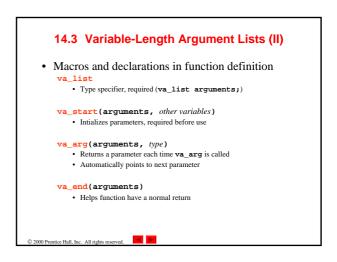
Chapter 14 - Advanced C Topics Outline 14.1 Introduction 14.2 Redirecting Input/Output on UNIX and DOS Systems 14.3 Variable-Length Argument Lists 14.4 Using Command-Line Arguments 14.5 Notes on Compiling Multiple-Source-File Programs 14.6 Program Termination with exit and atexit 14.7 The volatile Type Qualifier 14.8 Suffixes for Integer and Floating-Point Constants 14.9 More on Files 14.10 Signal Handling 14.11 Dynamic Memory Allocation with calloc and realloc 14.12 The Unconditional Branch: goto

14.1 Introduction Several advanced topics in this chapter Many capabilities are specific to operating systems (especially UNIX and/or DOS)

14.2 Redirecting Input/Output on UNIX and DOS Systems • Standard I/O - keyboard and screen - Redirect input and output • Redirect symbol (<) - Operating system feature, NOT C++ feature - UNIX and DOS - \$ or \$ represents command line Example: \$ myProgram < input - Rather than inputting values by hand, read them from a file • Pipe command (|) - Output of one program becomes input of another \$ firstProgram | secondProgram - Output of firstProgram goes to secondProgram

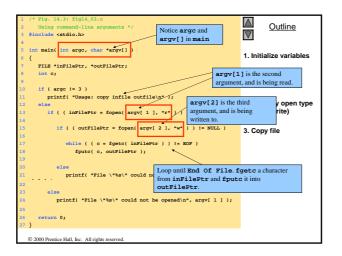
14.2 Redirecting Input/Output on UNIX and DOS Systems (II) Redirect output (>) Determines where output of a program goes \$\frac{\pinyProgram}{\pinyFile} \text{ myFile} \text{ (erases previous contents)} Append output (>>) Add output to end of file (preserve previous contents) \$\frac{\pinyOtherProgram}{\pinyOtherProgram} \text{ myFile} Output goes to the end of myFile}

14.3 Variable-Length Argument Lists • Functions with unspecified number of arguments - Load <stdarg.h> - Use ellipsis (...) at end of parameter list - Need at least one defined parameter double myfunction (int i, ...); - Prototype with variable length argument list - Example: prototype of printf int printf(const char*format, ...);



```
• Pass arguments to main in DOS and UNIX
int main( int argc, char *argv[] )
int argc - number of arguments passed
char *argv[] - array of strings, has names of arguments in
order (argv[ 0 ] is first argument)

Example: $ copy input output
argc: 3
argv[ 0 ]: "copy"
argv[ 1 ]: "input"
argv[ 2 ]: "output"
```



14.5 Notes on Compiling Multiple-Source-File Programs • Programs with multiple source files - Function definition must be in one file (cannot be split up) - Global variables accessible to functions in same file • Global variables must be defined in every file they are used - Example: • Integer myGlobal defined in one file • To use in another file: extern int myGlobal; • extern - states that variable defined elsewhere (i.e., not in that file)

14.5 Notes on Compiling Multiple-Source-File Programs (II)

- Programs with multiple source files (continued)
 - Function prototypes can be used in other files, extern not needed
 - · Have a prototype in each file that uses the function
 - Example: loading header files
 - #include <cstring>
 - · Contains prototypes of functions
 - We do not know where definitions are

14.5 Notes on Compiling Multiple-Source-File Programs (III)

- Keyword static
 - Variables can only be used in the file they are defined
- Programs with multiple source files
 - Tedious to compile everything if small changes made to one
 - Can recompile only the changed files
 - Procedure varies on system
 - UNIX: make utility

14.11 Dynamic Memory Allocation with calloc and realloc

- Dynamic memory allocation
 - Can create dynamic arrays
- void *calloc(size_t nmemb, size_t size)
 - nmemb number of memberssize size of each member

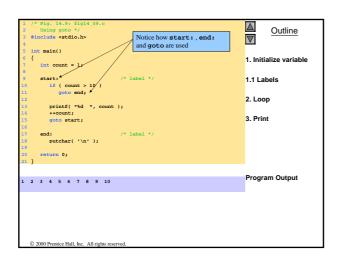
 - Returns pointer to dynamic array
- void *realloc(void *ptr, size_t size)
 - ptr pointer to the object being reallocated
 size new size of the object

 - Returns pointer to reallocated memory Returns NULL if cannot allocate space If newSize = 0, object freed

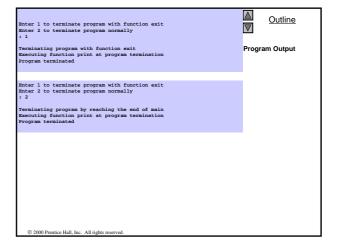
 - If pointerToObject = 0, acts like malloc

14.12 The Unconditional Branch: goto

- · Unstructured programming
 - Use when performance crucial
 - break to exit loop instead of waiting until condition becomes false
- goto statement
 - Changes flow control to first statement after specified label
 - Label: identifier and colon (i.e. start:)
 - Quick escape from deeply nested loop
 - goto start;



14.6 Program Termination with exit and atexit • Function exit - Forces a program to terminate - Parameters - symbolic constants EXIT_SUCCESS or EXIT_FAILURE - Returns implementation-defined value - exit(EXIT_SUCCESS); • Function atexit - atexit(functionToRun); - Registers functionToRun to execute upon successful program termination • atexit itself does not terminate the program - Register up to 32 functions (multiple atexit() statements) - Functions called in reverse register order - Called function cannot take arguments or return values



14.7 The volatile Type Qualifier

- volatile qualifier
 - Variable may be altered outside program
 - Variable not under control of program
 - Variable cannot be optimized

14.8 Suffixes for Integer and Floating-Point **Constants**

• C++ provides suffixes for constants.

Integer - u or U (unsigned integer)

long integer - 1 or L

unsigned long integer - ul or UL

float - for F

long double - 1 or LExamples: 174u

467L

3451ul

- Defaults
 - Integers: lowest type that holds them (int, long int, unsigned long int)
 - Floating point numbers: double

14.9 More on Files

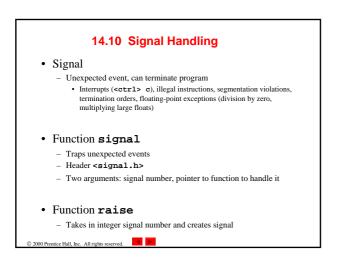
- C can process binary files

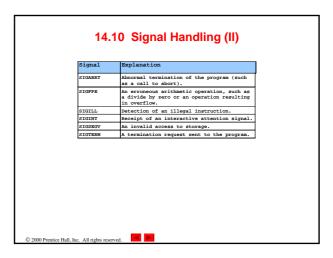
 - Not all systems support binary files
 Files opened as text files if binary mode not supported
 Binary files should be when rigorous speed, storage, and compatibility conditions demand it
 - Otherwise, text files preferred
 - · Inherent portability, can use standard tools to examine data
 - File open modes:

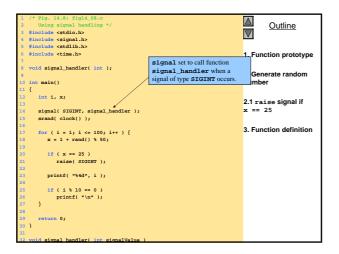
Mode	Description
rb	Open a binary file for reading.
wb	Create a binary file for writing. If the file already exists, discard the current contents.
ab	Append; open or create a binary file for writing at end-of-file.
rb+	Open a binary file for update (reading and writing).
wb+	Create a binary file for update. If the file already exists, discard the current contents.
ab+	Append; open or create a binary file for update; all writing is done at the end of the file

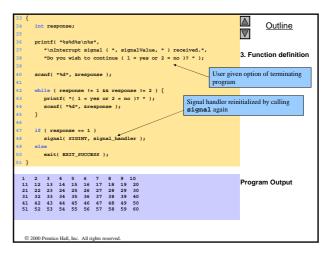
14.9 More on Files (II)

- Function tmpfile
 - In standard library
 - Opens a temporary file in mode "wb+"
 - some systems may process temporary files as text files
 - Temporary file exists until closed with fclose or until program
- Function rewind
 - Positions file pointers to the beginning of the file









```
61 62 63 64 65 65 67 68 69 70
71 72 73 74 75 76 77 78 79 80
91 92 93
91 92 93
Interrupt signal (2) received.
Do you wish to continue (1 = yes or 2 = no )? 1
94 95 96
Interrupt signal (2) received.
Do you wish to continue (1 = yes or 2 = no )? 1
97 98 99 100
```