

cis15-fall2007-sklar-lecIII.1

C string library (1).

• to use the string library, include the header in your C source file:

#include <string.h>

• string length function:

int strlen(char *s);

this function returns the number of characters in s; note that this is NOT the same thing as the number of characters allocated for the string array

• string comparison function:

int strcmp(const char *s1, const char *s2);

"This function returns an integer greater than, equal to, or less than 0, if the string pointed to by s1 is greater than, equal to, or less than the string pointed to by s2 respectively. The sign of a non-zero return value is determined by the sign of the difference between the values of the first pair of bytes that differ in the strings being compared."

• for more information and more string functions, do (e.g.):

unix\$ man strcmp

cis15-fall2007-sklar-lecIII.1

C string library (3).

• search functions:

char *strchr(const char *source, const char ch);

 $-\ensuremath{\,\mbox{returns}}$ pointer to first occurrence of ch in source; NULL if none

char *strstr(const char *source, const char *search);

- return pointer to first occurrence of search in source

```
C string library (2).
```

• copying functions:

char *strcpy(char *dest, char *source);

 $-\ensuremath{$ copies characters from source array into dest array up to NULL

char *strncpy(char *dest, char *source, int num);

 copies characters from source array into dest array; stops after num characters (if no NULL before that); appends NULL

cis15-fall2007-sklar-lecIII.1

C string library (4).

• parsing function:

char *strtok(char *s1, const char *s2);

- breaks string s1 into a series of *tokens*, delimited by s2

- called the first time with s1 equal to the string you want to break up

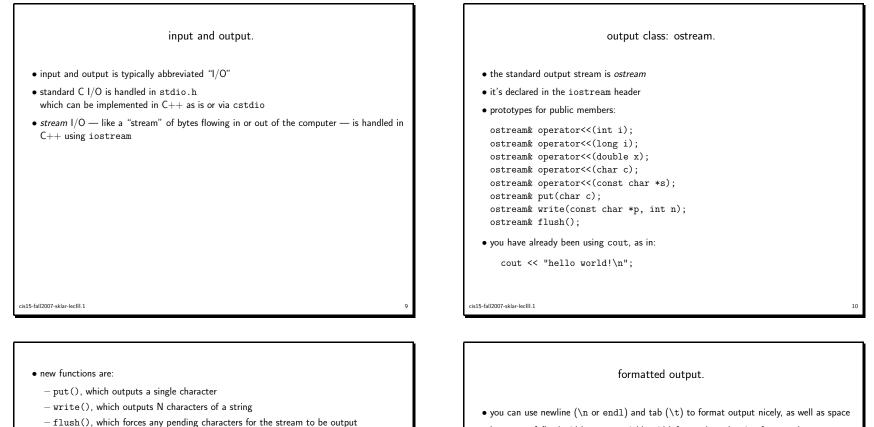
- called subsequent times with NULL as the first argument

- each time is called, it returns the next token on the string
- returns null when no more tokens remain

char inputline[1024];

char *name, *rank, *serial_num; printf("enter name+rank+serial number: "); scanf("%s", inputline); name = strtok(inputline,"+"); rank = strtok(null,"+"); serial_num = strtok(null,"+");

cis15-fall2007-sklar-lecIII.1



- be aware of *fixed width* versus *variable width* fonts when planning formatted output...
- there are some formatting functions in the ostream class: setf(), precision(), width()
- C++ also has a set of "manipulator" functions in iomanip
- some public functions:
 - scientific, which prints numbers using scientific notation
 - $-\operatorname{\tt left}$, which left justifies output
 - $-\operatorname{\tt right}$, which right justifies output
 - $-\, \texttt{setw}($ int), which sets the width of the output field
 - setfill(int), which sets the "fill" character
 - setbase(int), which sets the base format
- setprecision(int), which sets floating point precision
- cis15-fall2007-sklar-lecIII.1

formatted	output:	example	1.
-----------	---------	---------	----

#include <iostream>
#include <cmath>
using namespace std;

int main() {
 const int A = 5;
 const double B = 3.4568;
 double C;
 cout << "output using fixed precision, 2 decimal places:\n";
 cout.setf(ios::fixed, ios::floatfield);
 cout.precision(2);
 cout << "B=" << B << endl;
 cout << "output using width=10, left justified:\n";
 cout.setf(ios::left);
 cout.width(10);
 cout << "B=" << B << endl;
 cout << "output using width=10, right justified:\n";
</pre>

cout.setf(ios::right); cout.width(10); cout << "B=" << B << endl; cout << "you have to repeat the formatting if you want the same thing again: \: C = sin(B): cout.setf(ios::right); cout.width(10); cout << "C=" << C << endl; } // end of main() SAMPLE OUPUT: output using fixed precision, 2 decimal places: B=3.46 output using width=10, left justified: B= 3.46 output using width=10, right justified: B=3.46 you have to repeat the formatting if you want the same thing again: C=-0.31

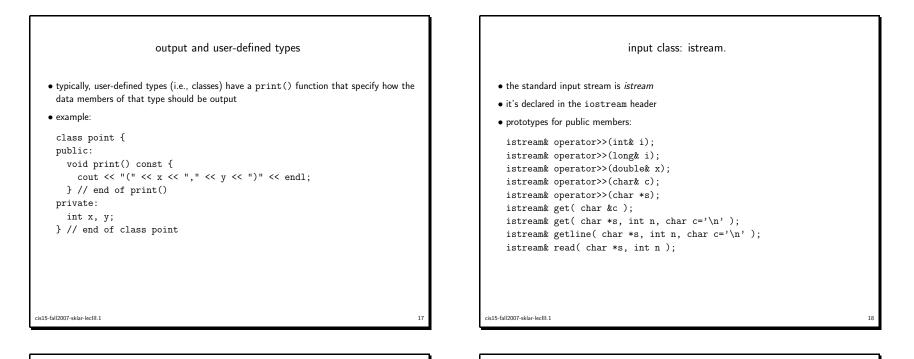
```
cis15-fall2007-sklar-lecIII.1
```

formatted output: example 2. #include <iostream> #include <iomanip> using namespace std; int main() { long double r; cout << "Enter radius: ";</pre> cin >> r;cout << "no formatting:</pre> area=" << r*r << endl; cout << "width:</pre> area=" << setw(20) << r*r << endl: cout << "width and precision: area="</pre> << setw(20) << setprecision(10) << r*r << endl; cout << "width, precision, fill: area=" << setfill('*')</pre> << setw(20) << setprecision(10) << r*r << endl; } // end of main()

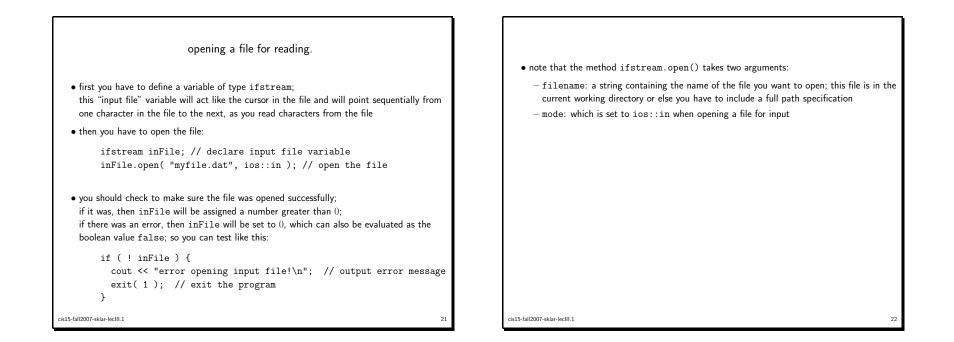
SAMPLE OUTPUT:

cis15-fall2007-sklar-lecIII.1

Enter radius: 34	area=1156	
no formatting: width:	area=	1156
vidth and precision:	area=	1156
width, precision, fill:	area=***************	*1156



• you have already been using cin, as in: files. int i: cout << "enter a number: ";</pre> • file handling involves three steps: cin >> i: 1. opening the file (for reading or writing) • new functions are: 2. reading from or writing to the file -get(), which reads in either a single character or a string of specified length 3. closing the file - getline(), which reads in a line (string) of specified length • files in C++ are *sequential access* - read(), which also reads in a string of specified length • think of a cursor that sits at a position in the file; • the functions that have n as a parameter, read in n-1 characters from the keyboard and with each read and write operation, you move that cursor's position in the file put a NULL (\0) string termination character in the n-th position • the last position in the file is called the "end-of-file", which is typically abbreviated as eof • the functions that have char c=' n' as a parameter, read until the specified *delimiter* is • all the functions described on the next few slides are defined in the either the <ifstream> read in: header file (for files you want to read from) or the <ofstream> header file (for files you the examples here use newline (n), but any character is okay to use want to write to) cis15-fall2007-sklar-lecIII.1 cis15-fall2007-sklar-lecIII.1



reading from a file.

```
• once the file is open, you can read from it
```

- you read from it in almost the same way that you read from the keyboard
- when you read from the keyboard, you use cin >> ...
- when you read from your input file, you use inFile >> ...
- here is an example:

```
int x, y;
inFile >> x;
inFile >> y;
```

```
• here is another example:
```

```
int x, y;
inFile >> x >> y;
```

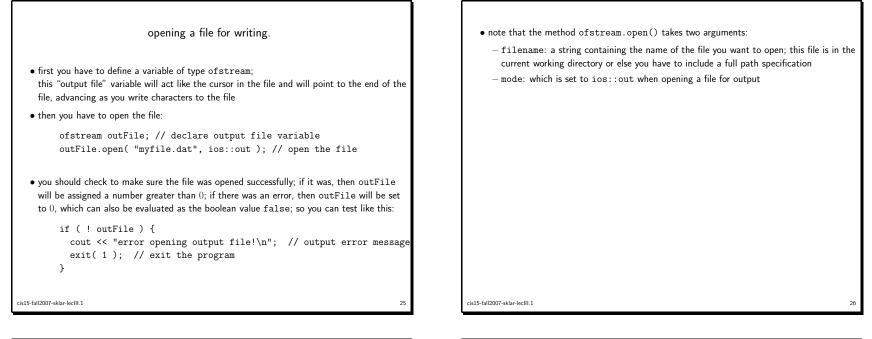
```
when reading from a file, you will need to check to make sure you have not read past the end of the file;
you do this by calling:
inFile.eof()
which will:

return true when you have gotten to the end of the file (i.e., read everything in the file)
return false when there is still something to read inside the file

for example:

while (! inFile.eof() ) {

inFile >> x;
cout << "x = " << x << endl;</li>
// end of while loop
```



writing to a file.

```
• once the file is open, you can write to it
```

- you write to it in almost the same way that you write to the screen
- \bullet when you write to the screen, you use cout << \ldots
- when you write to your output file, you use outFile << ...
- here is an example:

outFile << "hello world!\n";</pre>

• here is another example:

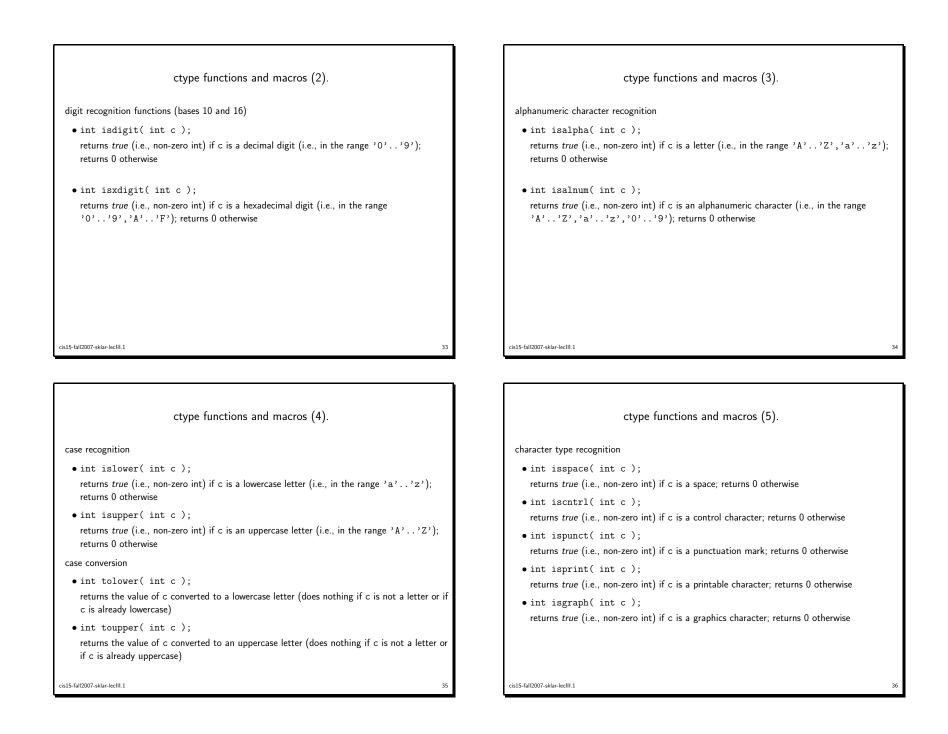
cis15-fall2007-sklar-lecIII.1

int x; outFile << "x = " << x << endl;</pre> closing a file.
 when you are done reading from or writing to a file, you need to close the file
 you do this using the close() function, which is part of both ifstream and ofstream
 so, to close a file that you opened for reading, you have do this, e.g.:
 inFile.close(); // close input file
 and, to close a file that you opened for writing, you have do this, e.g.:
 outFile.close(); // close output file
 that's all!

using strings as streams.	
• you can also use a string as a stream (i.e., to write output to a string or read input from a string)	
 the sstream header contains two data types: ostringstream for output istringstream for input 	
• example:	
<pre>#include <iostream> #include <sstream> using namespace std;</sstream></iostream></pre>	
<pre>int main() {</pre>	
<pre>#define MAXBUF 10 char buf[MAXBUF];</pre>	
<pre>char c; istringstream instring("my test string");</pre>	
s15-fall2007-sklar-lecIII.1 29	

ostringstream outstring; ostringstream outstring2(buf,ios::app); // input is read from "instring" instring >> c; cout << "c=[" << c << "]\n"; // output is written to "outstring" and "outstring2" outstring << c;</pre> outstring << c;</pre> cout << "outstring=[" << outstring.str() << "]\n";</pre> strcpy(buf,""); outstring2 << 'A';</pre> outstring2 << 'B';</pre> outstring2 << 'C';</pre> outstring2 << "DEF";</pre> cout << "outstring2=[" << outstring2.str() << "]\n";</pre> } cis15-fall2007-sklar-lecIII.1

SAMPLE OUTPUT:	ctype functions and macros (1).
<pre>c=[m] outstring=[mm] outstring2=[ABCDEF]</pre>	 character handling library <pre>#include <ctype.h></ctype.h></pre> digit recognition functions (bases 10 and 16) <pre>e alphanumeric character recognition case recognition/conversion character type recognition character type recognition character type recognition these are all of the form: int isdigit(int c); where the argument c is declared as an int, but it is intepreted as a char so if c = '0' (i.e., the ASCII value '0', index=48), then the function returns <i>true</i> (non-zero int) but if c = 0 (i.e., the ASCII value NULL, index=0), then the function returns <i>false</i> (0)</pre>
cis15-fall2007-sklar-lecIII.1 31	cis15-fall2007-sklar-lecIII.1 32



C style I/O (1).		
<pre>• #include <stdio.h> OR #include <cstdio> using namespace std;</cstdio></stdio.h></pre>		
 int printf(const char *format,) formatted output to stdout formatting: 		
cis15-fall2007-sklar-lecllJ.1	37	

Г

conversion character	argument	description
с	char	prints a single character
d or i	int	prints an integer
u	int	prints an unsigned int
0	int	prints an integer in octal
x or X	int	prints an integer in hexadecimal
e or E	float or double	print in scientific notation
f	float or double	print floating point value
g or G	float or double	same as e,E,f, or f — whichever uses fewest characters
s	char*	print a string
р	void*	print a pointer
%	none	print the % character

cis15-fall2007-sklar-lecIII.1

<section-header><equation-block><equation-block><section-header><section-header><section-header><section-header>

C style I/O (3).

- int scanf(const char *format, ...) formatted output to stdout
- formatting:

conversion	argument	description
character		
с	char*	reads a single character
d	int*	reads a decimal integer
i	int*	reads an integer in decimal,
		octal (leading 0) or hex (leading 0x)
u	int*	reads an unsigned int
0	int*	reads an integer in octal
x or X	int*	reads an integer in hexadecimal
e, E, f, F, g or G	float or double	reads a floating point value
S	char*	reads a string
р	void**	reads a pointer

 \bullet note that there is also <code>sscanf()</code>, which is like the C++ <code>istringstream</code> where you can read input from string

40

