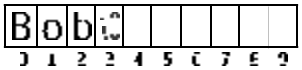
Topic: Character-based data

- The data type is *char*
- C-style strings (*string.h*)
- Type conversions
- The standard C++ class *string*
- File-stream input/output (*fstream.h*)
- C-language input/output (*stdio.h*)



Using *char* variables

[*charvar.cpp*]

```
#include <iostream>
using namespace std;
int main()
{
  char c1 = '.', c2 = 'z', c3 = '0';
  cout << "Enter 3 characters: ";
  cin >> c1 >> c2 >> c3;
  cout << "You entered "
     << c1 << c2 << c3 << endl;
  return EXIT_SUCCESS;
}
```

```
Enter 3 characters: DEF
You entered DEF
```


char as a character data type

- The ASCII table maps a character set to integers 0…127 (7 bits in 1 byte)
- Character literals use single quotes
- **`char c;`** declares a variable that stores *one* character
- **`cout << c;`** displays a character value *c*
- Standard character-related library: *ctype.h* (*toupper*, *tolower*, *isdigit*, *isalpha*, *ispunct*, *isprint*)

Using *toupper; char* variables

```
#include <ctype.h>
void main()
{
  char initial;
  cout << "Your initial? ";
  cin >> initial;
  char ucinit = toupper(initial);
  cout << "Hello, " << ucinit << endl;
}
```

standard **ctype.h** *function that returns* **int**

Sample I/O:

```
Your initial? d
Hello, D
```

[*initial.cpp*]

- Other <*ctype.h*> functions: *tolower*, *isalpha*, *isdigit*, *isalnum*, *ispunct*, *isprint*

Character / integer conversion

[*ascii.cpp*]

```
void main()
{
  char ch;
  cout << "Enter a character: ";
  cin >> ch;
  cout << "The ASCII code for " << ch
     << " is " << (int)ch << endl;
}
```

type cast

```
Enter a character: A
The ASCII code for A is 65
```

- *(int) ch* casts a *char* value to type *int*

int-to-*char* conversion

```
void main()
{
   int input;
   cout << "Enter ASCII value: "
   cin >> input;
   cout << "The character with ASCII "
      << input << " is " << char(input)
      << endl;
}
```

type cast converts numeric ASCII code to displayable character
may also be written (char) input

char constants

- Constants of *char* appear in single quotes:
  `'a' 'B' '$' '.' '"' ' '`
- Escape sequences express special characters:

carriage return	`'\n'`
tab	`'\t'`
backspace	`'\b'`
single quote	`'\''`
null character (ASCII 0)	`'\0'`
backslash	`'\\'`

C-style strings

- String literals use double quotes
- *char s[20];* declares a *char* array (string)
- Null character terminates a string
- Strings must be manipulated with libraries such as *string.h*
- Input: *cin >> s;*
- Output: *cout << s;*
- Initialization: *char s[] = "Foo";*
- Conversions: *atoi*, *atof*, (*stdlib.h*)

Accessing character elements of strings

[*greet.cpp*]

```
#include <ctype.h>
void main()
{
  cout << "Your first name? ";
  char name[20];
  cin >> name;
  name[0] = toupper(name[0]);
  cout << "Hello, " << name
    << "." << endl;
}
```

String variable declaration

Dimension (size)

Subscript

```
Your first name? jill
Hello, Jill.
```

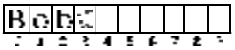

Valid and invalid string operations

Declare a 10-character string:

```
char name[10];
```

Valid:

```
strcpy(name,"Bob");
name[0] = 'J';
```



Syntax error (array address is constant):

~~`name = "Bob";`~~

Logic error (invalid subscript):

~~`name[10] = 's';`~~

Initialize a string:

```
const char NAME[10] = "Joan";
```


Some functions in the standard C library *string.h*

strlen(s): returns length of string parameter *s*

strcpy(s1,s2): copies contents of *s2* to *s1*

strcat(s1,s2): copies contents of *s2* to end of *s1*

strcmp(s1,s2): returns 0 if *s1* and *s2* are the same string; otherwise finds first character location x where *s1* and *s2* differ and returns (*s1[x]- s2[x]*)

Using the *string.h* library

[*strcat.cpp*]

```
#include <string.h>
void main()
{
  char first[40],last[40],fullname[80];
  cout << "Your first, last names? ";
  cin >> first >> last;
  strcpy(fullname,first);
  strcat(fullname," ");
  strcat(fullname,last);
  cout << "Hello, " << fullname << endl;
}
```

destination

source

Sample I/O:

```
Your first, last names? kris bernard
Hello, kris bernard
```

A safer string class

- C-style strings and arrays — error prone, require use of functions such as *strcpy* rather than operators such as '='
- A solution: standard ANSI/ISO class *string* (*#include <string>*)
- Some operators:
  << >> = += < == +
- Some member functions:
  at() length() substr()

C and C++ strings

	C-style	C++-style
Library	`<string.h>`	`<string>` `using namespace std;`
Declaration	`char s[80];`	`string s;`
Character	`char, s[0];`	`char, s[0];`
Assignment	`strcpy(s,"x");`	`s = "x";`
Concatenation	`strcat(s,"y");`	`s += "y";`
Length	`strlen(s)`	`s.length();`

The standard C++ *string* class

```
using namespace std;
#include <iostream>
#include <string>
using namespace std;
void main()
{
  string s1 = "Hello.txt";
  char s2[80] = s1.c_str();
  cout << "s1 = " << s1 << endl
     << "s2 = " << s2 << endl;
}
```

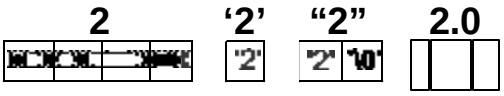
Automatic type conversion

Conversion function

Output:
```
Hello.txt
Hello.txt
```

[*ansi_str.cpp;* untested]

Storage for four standard types



int	*char*	*string*	*double*
(32 bits)	(1 byte)	(2 bytes for a 1-character string)	(64 bits in MSVC 5.0)

Stream input/output

- A stream is a sequence of characters coming from or going to a device
- *cin*, *cout* are stream objects
- Alternative destination/source for streams: strings; files
- We may use <<, >>, manipulators with any stream

Numeric to string conversions

[*numtostr.cpp*]

```
#include <string.h>
#include <strstrea.h>
void main()
{
  cout << "Enter an int and a float: ";
  int i;
  float f;
  cin >> i >> f;
  char s1[20], s2[20];
  ostrstream strout1(s1,20),strout2(s2,20);
  strout1 << i << ends;
  strout2 << f << ends;
  cout << s1 << " and " << s2
     << " are strings of lengths "
     << strlen(s1) << " and "
     << strlen(s2) << endl;
}
```

String stream library

String stream class

String stream objects

Store value of int **i** *in string* **s1** *via string stream*

Sample I/O:
```
Enter an int and a float: 2 4.5
2 and 4.5 are strings of lengths 1 and 3
```

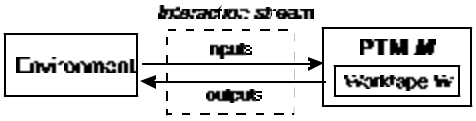
File stream input/output

- Standard header: *fstream.h*
- Types: *ofstream* (output to disk), *ifstream* (input from disk)
- Using these types we declare a stream object to use like *cin* or *cout:*

```
ofstream fout("myfile.txt");
fout << "Hello";
```

- File stream functions: *open( ), close( ), eof( ), bad( ), good( )*

A stream is infinite



- Stored data occupies finite space
- A user-controlled loop may never end
- I/O streams express *interactive* (non-algorithmic) computations

Reading 2 integers from a file (C++)

```
#include <iostream.h>
#include <fstream.h>
void main()
{
  cout << "Reading 2 integers from file"
       << "ADDINFIL.DAT." << endl;
  ifstream
  infile("ADDINFIL.DAT",ios::nocreate);
  if (infile.bad())
    cout << "File not found\n";
  else
  {
    int input1,input2;
    infile >> input1 >> input2;
    infile.close();
    cout << "In file: " << input1 << ", "
         << input2 << endl;
  }
}
```

C++ file stream library

Open file

Read file

[*addinfil.cpp*]

Writing to a file in C++

```
#include <fstream.h>

void main()
{
  cout << "Enter 2 integers to add: ";
  int input1,input2,sum;
  cin >> input1 >> input2;
  sum = input1 + input2;
  ofstream outfile("FILEADD.OUT");
  outfile << input1 << " + " << input2
      << " = " << sum << endl;
  outfile.close();
  cout << "The following was written to "
     "FILEADD.OUT:\n";
  cout << input1 << " + " << input2
      << " = " << sum << endl;
}
```

[*fileadd.cpp*]

C-language input/output example

```
#include <stdio.h>

void main()
{
  int age;
  printf("Age? ");
  scanf("%i", &age);
  printf("You are %i months old",12 * age);
}
```

- *printf* and *scanf* have:
  - format-string parameter, containing *type specifiers*
  - values to be inserted in place of type specs
- Type specs:
  %i, %d (int); %c (char); %f (float); %s (string)

C-style input/output

- Standard header: *stdio.h*
- Output: *printf("Hello, %i\n", ID_num);*
- Input: *scanf("%i", &ID_num);*

Format strings

- Parameters are a *format string* and a list of data items (*printf*) or variables (*scanf*)
- Format string may contain type specifiers for other parameters
- *scanf* needs an address for each input item
- File I/O tools: pointer-to-*FILE* type, *fopen*, *fclose*, *fprintf*, *fscanf*, others

Using C type specifiers

```
int qty = 4;
double price = 59.95;
printf("Qty = %i   price = %2.2f\n",
     qty, price);
```

Specifies floating-point value formatted with 2 digits before, after decimal point

Specifies int value

C and C++ input/output

	C++	C
Console	<iostream.h> <iomanip.h>	<stdio.h>
input	*cin >> x;*	*scanf("%i",&x);*
output	*cout << x << endl;*	*printf("%i\n",x);*
File	<fstream.h>	<stdio.h>
input	*ifstream fin("IN");* *fin >> x;*	*FILE* is = fopen("IN","r");* *fscanf(is,"%i",&x);*
output	*ofstream fout ("OUT");* *fout << x << endl;*	*FILE* os = fopen("OUT","w");* *fprintf(os,"%i",x);*
Newline	*cout << "Bye" << endl; printf("Bye\n");*	

C vs. C++ streams

- In C you must be sure to use accurate type specifiers (*%d, %i, %f, %s, %c*)
- C *printf* type spec is error prone:

```
double rate = 32.4;
printf("%d",rate);
```

Output:
858993459

- In C++, the << and >> may be overloaded so that they perform I/O for any type; no type spec needed
- << and >> work with console, file, or string stream I/O

Ways to categorize data types

- *Standard vs. programmer-defined*
- All types seen so far are standard
- *Simple vs. compound*
- Most types seen so far are simple; strings, structures and arrays are compound
- *Pointer vs. non-pointer*
- The type of *&n* as in *scanf("%i",&n);* is pointer

Memory allocation for instances of different types

```
#include <iostream.h>

void main()
{
 char s[80] = "Zoom";
 cout << "Type      Size" << endl
   << "int       " << sizeof(int) << endl
   << "float     " << sizeof(float) << endl
   << "double    " << sizeof(double) << endl
   << "char       " << sizeof(char) << endl
   << "C string   " << sizeof(s) << endl
}
```

```
...Size
   ...4
   ...4
   ...8
   ...1
  ...80
```

[*alloc.cpp*]

Line input

- To input a string that may contain spaces:

```
char line[80];
cin.getline(line,79);
```

- *Note:* Mixing line and numeric input should be avoided.
- Alternative string input function in <stdio.h> that accepts spaces:

```
gets(last);
```

- To center output:

```
cout << setw((80-strlen(line1)) / 2)
   << "" << line1 << endl
cout << setw((80-strlen(line2)) / 2)
   << "" << line2 << endl;
```