Fundamental Data Types

Lecture 4

Sections 2.7 - 2.10

Robb T. Koether

Hampden-Sydney College

Mon, Sep 2, 2019

Outline

Integer Types

Integer Type

```
cout << 123 << endl;
```

- Integers are stored as binary numbers.
- An n-bit integer can hold any of 2^n different values.
- Integer types are either signed or unsigned.
- Integer literals must not have a decimal point and are signed integers by default.
- To force a literal to be unsigned, we must cast it as such:

```
unsigned int(123)
```

Integer Types

Integer Types

```
int a = 123;
int b = -456;
unsigned int c = 789;
```

- The integer types.
  - **short** – 2 bytes
  - **unsigned short** – 2 bytes
  - **int** – 4 bytes
  - **unsigned int** – 4 bytes
  - **long** – 4 bytes
  - **unsigned long** – 4 bytes
- Each type can be either signed or unsigned.
- On some computers, a **long** can be 8 bytes.

Signed vs. Unsigned Integers

- Unsigned integers.
  - An unsigned integer cannot be negative.
  - Values range from 0 to $2^{32} - 1$.
- Signed integers
  - A signed integer can be positive or negative.
  - The high-order bit (sign bit) indicates the sign.
  - Values range from -2^{31} to $2^{31} - 1$.
- By default, integers are signed.

Example of Signed and Unsigned Integers

Binary	Unsigned	Signed
00000000	0	0
00000001	1	1
00000010	2	2
00000011	3	3
⋮	⋮	⋮
01111111	127	127
10000000	128	-128
10000001	129	-127
10000010	130	-126
10000011	131	-125
⋮	⋮	⋮
11111111	255	-1

8-bit integers, unsigned and signed

Example of Signed and Unsigned Integers

Binary	Unsigned	Signed
0000000000000000	0	0
0000000000000001	1	1
0000000000000010	2	2
0000000000000011	3	3
⋮	⋮	⋮
0111111111111111	32767	32767
1000000000000000	32768	-32768
1000000000000001	32769	-32767
1000000000000010	32770	-32766
1000000000000011	32771	-32765
⋮	⋮	⋮
1111111111111111	65535	-1

16-bit **short**, unsigned and signed

Example of Signed and Unsigned Integers

Binary	Unsigned	Signed
00000000000000000000000000000000	0	0
00000000000000000000000000000001	1	1
00000000000000000000000000000010	2	2
00000000000000000000000000000011	3	3
⋮	⋮	⋮
01111111111111111111111111111111	2147483647	2147483647
10000000000000000000000000000000	2147483648	-2147483648
10000000000000000000000000000001	2147483649	-2147483647
10000000000000000000000000000010	2147483650	-2147483646
10000000000000000000000000000011	2147483651	-2147483645
⋮	⋮	⋮
11111111111111111111111111111111	4294967295	-1

32-bit **int**, unsigned and signed

Ranges of Values of Integer Type

Type	Range	
	From	To
`unsigned short`	0	65535
`short`	$-32,768$	32,767
`unsigned int`	0	4,294,967,295
`int`	$-2,147,483,648$	2,147,483,647

Integer Overflow

- What happens when an integer value becomes too large?
- That is, what if we assign to an integer the largest legal value, and then add 1?
- Example
  - `IntLimitTest.cpp`

Outline

Floating-Point Types

Floating-Point

```
float a = 123.456f;
double b = 123.456789012345;
```

- The floating-point types.
  - **`float`** – 4 bytes
  - **`double`** – 8 bytes
  - **`long double`** – 8 bytes

Floating-Point Types

Floating-Point Type

```
cout << 12.34 << endl;
```

- Floating-point numbers are stored in three parts: sign bit, exponent, and mantissa.
- **`float`**
  - The sign bit (0 = +, 1 = −).
  - An 8-bit exponent locates the decimal point.
  - A 23-bit mantissa contains the significant digits.
- **`double`**
  - The sign bit (0 = +, 1 = −).
  - An 11-bit exponent locates the decimal point.
  - A 52-bit mantissa contains the significant digits.
- Floating-point literals must have a decimal point and are doubles by default.

Floating-Point Literals

- Floating-point literals must have a decimal point and are **`doubles`** by default.
- To force a floating-point literal to be a **`float`**, we may cast is

```
float(12.34)
```

or we may append the letter `f`:

```
12.34f
```

Single-Precision Numbers

`float` Type

```
float x = 12.34567f;
```

- The positive values of a **`float`** range from a minimum of $\pm 1.17549 \times 10^{-38}$ to a maximum of $\pm 3.40282 \times 10^{38}$.
- Approximately 7 decimal-digit precision.

Double-Precision Numbers

`double` Type

```
double pi = 3.141592653589793;
```

- The positive values of a **`double`** range from a minimum of $\pm 2.22507 \times 10^{-308}$ to a maximum of $\pm 1.79769 \times 10^{308}$.
- Approximately 16 decimal-digit precision.

Floating-Point Overflow and Underflow

- What happens if we begin with the largest possible **`float`** and then double it?
- What happens if we begin with the smallest possible positive **`float`** and divide it by 2?
- Example
  - `FloatLimitTest.cpp`

Outline

The Character Type

`char` Type

```
char letter = 'a';
```

- Characters (**`char`**s) are stored as one-byte integers using the ASCII values (see p. 106).
- A character can have any of 256 different values.
- Character literals must use *single* quotation marks, e.g., `'A'`.

ASCII Table (0 - 63)

ASCII	Char
0	NULL
1	SOTT
2	STX
3	ETY
4	EOT
5	ENQ
6	ACK
7	BELL
8	BKSPC
9	HZTAB
10	NEWLN
11	VTAB
12	FF
13	CR
14	SO
15	SI

ASCII	Char
16	DLE
17	DC1
18	DC2
19	DC3
20	DC4
21	NAK
22	SYN
23	ETB
24	CAN
25	EM
26	SUB
27	ESC
28	FS
29	GS
30	RS
31	US

ASCII	Char
32	(space)
33	!
34	"
35	#
36	$
37	%
38	&
39	‘
40	(
41	)
42	*
43	+
44	’
45	_
46	.
47	/

ASCII	Char
48	0
49	1
50	2
51	3
52	4
53	5
54	6
55	7
56	8
57	9
58	:
59	;
60	<
61	=
62	>
63	?

ASCII Table (64 - 127)

ASCII	Char
64	@
65	A
66	B
67	C
68	D
69	E
70	F
71	G
72	H
73	I
74	J
75	K
76	L
77	M
78	N
79	O

ASCII	Char
80	P
81	Q
82	R
83	S
84	T
85	U
86	V
87	W
88	X
89	Y
90	Z
91	[
92	\
93	]
94	^
95	_

ASCII	Char
96	‘
97	a
98	b
99	c
100	d
101	e
102	f
103	g
104	h
105	i
106	j
107	k
108	l
109	m
110	n
111	o

ASCII	Char
112	p
113	q
114	r
115	s
116	t
117	u
118	v
119	w
120	x
121	y
122	z
123	{
124	|
125	}
126	~
127	DEL

Characters as Integers

`char` Type

```
char letter = 'a';
int value = letter;
letter = value + 1;
```

- Characters are interchangeable with integers in the range 0 to 255.
- The numerical value of a character is its ASCII value.
  - Blank space (ASCII 32).
  - Digits 0 - 9 (ASCII 48 - 57).
  - Uppercase letters A - Z (ASCII 65 - 90).
  - Lowercase letters a - z (ASCII 97 - 122).
- Characters are ordered according to their ASCII values.

Outline

The `string` Type

`string` Type

```
string message = "Hello, World";
```

- A string is stored as a sequence of characters.
- A string may hold any number of characters, including none.
- String literals must use *double* quotation marks, e.g., `"Hello"`.

The `string` Type

`string` Type

```
string msg = "Hello, World";
cout << msg[9] << msg[1] << msg[11] << endl;
```

- The characters in the string are indexed, beginning with index 0.
- They can be accessed individually by writing the index within square brackets `[...]`.

Calculations with Characters

- Example
  - `CharCalcs.cpp`

Outline

Assignment

Assignment

- Read Sections 2.7 - 2.10.