

Topic: Numeric data types

- The *int* data type
- Arithmetic expressions and operators
- Smaller and larger-capacity integer types
- Using floating-point data
- Standard numeric constants and functions
- Type conversions
- Formatting output

David Keil 9/03 1

The data type *int*

- Data type: a category that defines the storage and meaning or interpretation given to a pattern of bits
- *int* is a signed integer type
- Usual range of values: $-2G \dots 2G$ (32 bits: $2^32 = 4G$)
- 2586 is an *expression* of type *int*
- After *int* $x = 9$;, x is an expression of type *int*

David Keil 9/03 2

Arithmetic operators

- *Binary operators have 2 operands:*
+ - * /
(Note: division of integers produces an *int* value; division by 0 is undefined)
- % modulo or remainder operator; e.g.:
- $7 \% 3 = 1$
- The clock time 3 hours after 11:00 is $(11 + 3) \% 12$
- The clock time h hours after time t is $(t + h) \% 12$, except a result of 0 should change to 12
- *Unary negation:* -2 - y - $(5 + 9)$

David Keil 9/03 3

Converting cents to dollars

```
cout << "Cents: ";
int cents;
cin >> cents;
int dollars = cents / 100;
int change = cents % 100;
cout << dollars << " dollars, "
    << change << "cents" << endl;
```

```
Cents: 235
2 dollars, 35 cents
```

- integer division yields an integer
- the modulo operator can truncate a number

David Keil 9/03 4

Math notation and C/C++

Math	C/C++
$\frac{a+1}{b-2}$	(a+1) / (b-2)
a^2	a * a or pow(a, 2)
c modulo d	c % d
$\sqrt{2}$	sqrt(2)

David Keil 9/03 5

Extended assignment operators

- $x += y$ means $x = x + y$
- $x -= y$ means $x = x - y$
- $x *= y$ means $x = x * y$
- $x /= y$ means $x = x / y$
- $x \% = y$ means $x = x \% y$
- $++x$ or $x++$ mean $x = x + 1$
- $--x$ or $x--$ mean $x = x - 1$

David Keil 9/03 6

Pre- and post-increment operators

```
int n = 4;
cout << "n++ = " << n++ << endl
     << "now n = " << n << endl
     << "++n = " << ++n << endl
     << "n = " << n << endl;
```

```
n++ = 4
now n = 5
++n = 6
n = 6
```

- Both operators add 1 to value of variable
- Pre-increment operates before evaluation

David Keil 9/03 7

Operator precedence

- Parenthesized operations come first
- Unary minus has high precedence
- Multiplication and division precede addition and subtraction
- Operations of same precedence proceed left to right

• *Examples:*

```
8 - 2 + 5      8 - (2 + 5)
3 * 2 + 4      3 * (2 + 4)  3 + 2 * 4
3 + 6 / 2      (3 + 6) / 2  -2 + 3
1 + 3 % 2      25 % 5 * 2
```

David Keil 9/03 8

Small and large integer types

Type	Storage (bytes*)	Min. value*	Max. value*
<i>char</i>	1	0	255
<i>int</i>	4	-2G	+2G
<i>unsigned int</i>	4	0	+4G
<i>short int</i>	2	-32K	+32K
<i>long int</i>	4	-2G	+2G

- Type qualifiers: *short*, *long*, *unsigned*, *const*
- Use the *sizeof* operator; e.g.:

```
cout << sizeof (char);
```

Output: 1

*Compiler dependent; figures here are from MSVC Ver. 5.0

David Keil 9/03 9

Data types *float* and *double*

- May store fractional values and a great range of numbers
- Storage: sign bit, fraction, exponent, based on scientific-notation concept
- *float* occupies 32 bits in MSVC 5.0 implementation
- Floating-point storage entails representational error
- Qualifier *double* extends precision (64 bits in MSVC)

David Keil 9/03 10

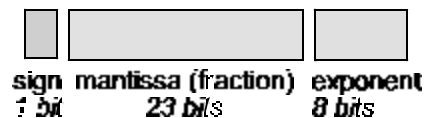
Using *double*

```
cout << "Enter 2 #s: ";
double a,b;
cin >> a >> b;
double quotient = a / b;
cout << "quotient = "
     << quotient << endl;
```

```
Enter 2 #s: 1 7
quotient = 0.142857
```

- Result of dividing floating-point values is of floating-point type

David Keil 9/03 11

Floating-point data

- Data types *float* and *double* represent numbers with possible fraction parts
- The numeral floats because the exponent part compensates for a shift to eliminate 0's on the left
- *double* has twice the precision of *float*

David Keil 9/03 12

Standard numeric identifiers

- Header file *math.h* (`#include <math.h>`):
 - absolute value: *abs*, *fabs*
 - trigonometry: *cos*, *acos*, *sin*, *asin*, *tan*, *atan*
 - other: *exp*, *log*, *pow*, *sqrt*

- Example:

```
void main()
{
  cout << abs(-3.5) << endl
    << fabs(-3.5) << endl
    << sqrt(2) << endl
    << log(16) / log(2) << endl
    << pow(3,4) << endl
    << cos(3.1416 / 360) << endl;
}
```

[math.cpp] David Keil 9/03 13

Output:

```
3
3.5
1.41421
4
81
0.999962
```

Numeric type conversions

- Automatic: occurs by promotion or truncation. *Examples:*

```
int y = 4.3; cout << y;
double z = 2;
```
- High types to low: *double, float, long, int, short, char*
- Type casts coerce types.
 - E.g., `(double(1) / 2)` yields 0.5
 - `cout << (int) 4.2;` outputs 4
- Danger: overflow (watch warnings)

David Keil 9/03 14

Type casts and integer division

```
void main()
{
  int a=5,b=3;
  cout << a << " / " << b << " = "
    << a / b << endl;
  cout << "double(" << a << " ) / " << b
    << " = " << double(a) / b << endl;
}
```

[intfloat.cpp]

Output:

```
5 / 3 = 1   double(5) / 3 = 1.66667
```

- Without type cast, integer division occurs here, yielding integer result
- Type casting forces a data item to have a specified type

David Keil 9/03 15

C++ promotes type for compatibility

- Lower type is promoted to higher
- High to low:
 - double, float, long int, int, char*

- Examples:

```
double f = 2;
double sum = 1 + 2.2;
(3.6 * 4)
```

promoted to double
type of this expression is double

David Keil 9/03 16

Integer overflow

```
void main()
{
  short int product = 2 * 20000;
  cout << "2 * 20000 = "
    << product << endl;
}
```

[overflow.cpp]

Output:

```
2 * 20000 = -25536
```

- Overflow occurs when a value assigned exceeds the capacity of a data type

David Keil 9/03 17

Operator overloading

- In C and C++, the meaning of some operators depends on the context
- Examples: `=`, `+`, `-`, `*`, `/`, `<<`, `>>`
- Recall that in machine language, different instructions are used for operations on different types of data

David Keil 9/03 18

Manipulators

- Library: *iomanip.h*
- Used to format stream output
- A manipulator changes the state of an output stream object
- Areas of control: field width, justification, precision
- Manipulator functions: *setw*, *setprecision*, *setiosflags*

David Keil 9/03 19

Manipulator *setw*: example

```
// intcol2.cpp
// Aligns integers on right.
#include <iostream.h>
#include <iomanip.h>
const int FW = 6; // field width
void main()
{
    int n1 = 32, n2 = 110, n3 = 7;
    cout << setw(FW) << n1 << endl
         << setw(FW) << n2 << endl
         << setw(FW) << n3 << endl;
}
```

Output:

				32
			110	
			7	

David Keil 9/03 20

Setting width and precision of fractions

```
#include <iostream.h>
#include <iomanip.h>

void main()
{
    double f = 18.558;
    cout << setprecision(2)
         << setiosflags(ios::showpoint | ios::fixed)
         << setw(6) << f << endl;
}
```

Output: 18.56

[floutout2.cpp]

David Keil 9/03 21

Discussion problems

- Write a program that prompts for a unit price, discount rate, and quantity and displays a subtotal without tax; a tax amount, given a constant tax rate of 5%; and a total amount due. Format to two places.
- Write a program that prompts for the height and diameter of an oil drum and displays the area of its surface, given that the area of a circle is $(\pi \times \text{radius}^2)$ and $\pi = 3.14159$. Format to two places.

David Keil 9/03 22

Formatting tabular output

```
char name1[80] = "Wang",
      name2[80] = "Fuentes";
int sal1 = 25000, sal2 = 37820;

cout << setw(12) << "Name" << setw(8)
     << "Salary" << endl;
cout << setw(12) << name1
     << setw(8) << sal1 << endl;
cout << setw(12) << name2
     << setw(8) << sal2 << endl;
```

Name	Salary
Wang	25000
Fuentes	37820

2 lines of table entries

David Keil 9/03 23

Using field width and precision

```
char name[80] = "book";
double price = 25.95, qty = 37;

cout << setprecision(2)
     << setiosflags(ios::showpoint | ios::fixed)
     << setw(12) << "Name"
     << setw(8) << "Price"
     << setw(8) << "Qty" << endl;
cout << setw(12) << name
     << setw(8) << price
     << setw(8) << qty << endl;
```

Name	Price	Qty
book	25.95	37

David Keil 9/03 24