C++ Basics
Part Two

Sai-Keung Wong

National Yang Ming Chiao Tung University

Hsinchu, Taiwan

Intended Learning Outcomes

- Describe how to perform a dry run
- Describe how to use vector
- List some elementary programming concepts

Tracing a program

Do step-by-step

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;              // declaration; local variable
  double width = 4;           // declaration and initialization; local variable

  // instruction 1:
  height = 6;                                          // assignment

// instruction 2: Compute area
  double area = height* width;                         // local variable

  // instruction 3: Display the area
  cout << "The rectangle area is “ <<  area << endl;
}
```

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;            // declaration
  double width = 4;         // declaration and initialization

  // instruction 1:
  height = 6;               // assignment

// instruction 2: Compute area
  double area = height* width;

  // instruction 3: Display the area
  cout << "The rectangle area is “ <<  area << endl;
}
```

height	???
width	???
area	???

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;            // declaration
  double width = 4;         // declaration and initialization

  // instruction 1:
  height = 6;               // assignment

// instruction 2: Compute area
  double area = height* width;

  // instruction 3: Display the area
  cout << "The rectangle area is “ <<  area << endl;
}
```

height	???
width	4
area	???

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;            // declaration. Local variable
  double width = 4;         // declaration and initialization. Local variable

  // instruction 1:
  height = 6;               // assignment

// instruction 2: Compute area
  double area = height* width;

  // instruction 3: Display the area
  cout << "The rectangle area is “ <<  area << endl;
}
```

height	6
width	4
area	????

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;          // declaration
  double width = 4;       // declaration and initialization

  // instruction 1:
  height = 6;             // assignment

// instruction 2: Compute area
  double area = height* width;

  // instruction 3: Display the area
  cout << "The rectangle area is “ <<  area << endl;
}
```

height	6
width	4
area	24

Declare area with initialization. Compute the product and store the result to area.

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;            // declaration
  double width = 4;         // declaration and initialization

  // instruction 1:
  height = 6;               // assignment

// instruction 2: Compute area
  double area = height* width;

  // instruction 3: Display the area
  cout << "The rectangle area is “ <<  area << endl;
}
```

Memory address

height	6
width	4
area	24

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;            // declaration
  double width = 4;         // declaration and initialization

  // instruction 1:
  height = 6;               // assignment

// instruction 2: Compute area
  double area = height* width;

  // instruction 3: Display the area
  cout << "The rectangle area is “ <<  area << endl;
}
```

		Memory address
height	6	FF000020
width	4	FF000018
area	24	FF000010

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;            // declaration
  double width = 4;         // declaration and initialization

  // instruction 1:
  height = 6;               // assignment

// instruction 2: Compute area
  double area = height* width;

  // instruction 3: Display the area
  cout << "The rectangle area is “ <<  area << endl;
}
```

Memory address

height	6	FF000020
width	4	FF000018
area	24	FF000010

Need to allocate A1 beforehand.
They are A2 stored in the run-time A3
Each variable has an A4

Tracing a program (Dry run)

```
#include <iostream>
using namespace std;
int main() {
  double height;            // declaration
  double width = 4;         // declaration and initialization

  // instruction 1:
  height = 6;               // assignment

// instruction 2: Compute area
  double area = height* width;

  // instruction 3: Display the area
  cout << "The rectangle area is " <<  area << endl;
}
```

height	6	FF000020
width	4	FF000018
area	24	FF000010

Print a message and display the value of *area*.

Reading Input from the Keyboard

```
cin >> variable;
```

Example:

```
int health_points;
cin >> health_points;

float fValue;
cin >> fValue;

unsigned int score;
cin >> score;

string str;         // include <string>
cin >> str;
```

Example: Write a program to read a set of integers and show them one by one

Example: Write a program to read a set of integers and show them one by one

Output:

8

5

3

1

-9

Example: Write a program to read a set of integers and show them one by one

std::vector

std::vector is a sequence container that encapsulates dynamic size arrays.

e.g.,

```
std::vector<int> a; // a is empty
a.push_back(4);
a.push_back(3);
a.push_back(5);
```

Example: Write a program to read a set of integers and show them one by one

std::vector

std::vector is a sequence container that encapsulates dynamic size arrays.

e.g.,

```
std::vector<int> a; // a is empty. L1
a.push_back(4);     // L2
a.push_back(3);     // L3
a.push_back(5);     // L4
```

The content of a

L1: a

L2: a

L3: a

L4: a

Example: Write a program to read a set of integers and show them one by one

std::vector

std::vector is a sequence container that encapsulates dynamic size arrays.

e.g.,

```
std::vector<int> a; // a is empty. L1
a.push_back(4);                 // L2
a.push_back(3);                 // L3
a.push_back(5);                 // L4
```

The content of a

L1: a

Example: Write a program to read a set of integers and show them one by one

std::vector

std::vector is a sequence container that encapsulates dynamic size arrays.

e.g.,

```
std::vector<int> a; // a is empty. L1
a.push_back(4);                // L2
a.push_back(3);                // L3
a.push_back(5);                // L4
```

The content of a

L1: a

L2: a 4

L3: a

L4: a

Example: Write a program to read a set of integers and show them one by one

std::vector

std::vector is a sequence container that encapsulates dynamic size arrays.

e.g.,

```
std::vector<int> a; // a is empty. L1
a.push_back(4);     // L2
a.push_back(3);     // L3
a.push_back(5);     // L4
```

The content of a

L1: a

L2: a 4

L3: a 4 3

L4: a

Example: Write a program to read a set of integers and show them one by one

std::vector

std::vector is a sequence container that encapsulates dynamic size arrays.

e.g.,

```
std::vector<int> a; // a is empty. L1
a.push_back(4);                 // L2
a.push_back(3);                 // L3
a.push_back(5);                 // L4
```

The content of a

L1: a

L2: a 4

L3: a 4 3

L4: a 4 3 5

Example: Write a program to read a set of integers and show them one by one

```
int main() {
   int num;
   cin >> num;
   std::vector<int> s;
   for ( int i = 0; i < num; ++i ) {
                int input;
                cin >> input;
                s.push_back( input );
   }
   for ( int i = 0; i < s.size( ); ++i )  {
                cout << s[ i ] << endl;
   }
   return 0;
}
```

Example: Write a program to read a set of integers and show them one by one

```
int main() {
   int num;
   cin >> num;
   std::vector<int> s;
   for ( int i = 0; i < num; ++i ) {
            int input;
            cin >> input;
            s.push_back( input );
   }
   for ( int i = 0; i < s.size( ); ++i )  {
            cout << s[ i ] << endl;
   }
   return 0;
}
```

```
void process( ) {
   int num;
   cin >> num;
   std::vector<int> s;
   for ( int i = 0; i < num; ++i ) {
            int input;
            cin >> input;
            s.push_back( input );
   }
   for ( int i = 0; i < s.size( ); ++i )  {
            cout << s[ i ] << endl;
   }
}
```

```
int main() {
   process( );
   return 0;
}
```

Example: Write a program to read a set of integers and show them one by one

```
int main() {
   int num;
   cin >> num;
   std::vector<int> s;
   for ( int i = 0; i < num; ++i ) {
            int input;
            cin >> input;
            s.push_back( input );
   }
   for ( int i = 0; i < s.size( ); ++i )  {
            cout << s[ i ] << endl;
   }
   return 0;
}
```

```
void process( ) {
   int num;
   cin >> num;
   std::vector<int> s;
   for ( int i = 0; i < num; ++i ) {
            int input;
            cin >> input;
            s.push_back( input );
   }
   for ( int i = 0; i < s.size( ); ++i )  {
            cout << s[ i ] << endl;
   }
}
```

```
// headers?



int main() {
   process( );
   return 0;
}
```

Example: Write a program to read a set of integers and show them one by one

```
int main() {
   int num;
   cin >> num;
   std::vector<int> s;
   for ( int i = 0; i < num; ++i ) {
            int input;
            cin >> input;
            s.push_back( input );
   }
   for ( int i = 0; i < s.size( ); ++i )  {
            cout << s[ i ] << endl;
   }
   return 0;
}
```

```
void process( ) {
   int num;
   cin >> num;
   std::vector<int> s;
   for ( int i = 0; i < num; ++i ) {
            int input;
            cin >> input;
            s.push_back( input );
   }
   for ( int i = 0; i < s.size( ); ++i )  {
            cout << s[ i ] << endl;
   }
}
```

```
// headers?
#include <iostream>
#include <vector>
using namespace std;
int main() {
   process( );
   return 0;
}
```

Example: Write a program to read a set of integers and show them one by one

```
void process( ) {
   int num;
   cin >> num;
   std::vector<int> s;
   for ( int i = 0; i < num; ++i ) {
            int input;
            cin >> input;
            s.push_back( input );
   }
   for ( int i = 0; i < s.size( ); ++i )  {
            cout << s[ i ] << endl;
   }
}
```

```
// headers?
#include <iostream>
#include <vector>
using namespace std;
int main() {
   process( );
   return 0;
}
```

What are the problems of this program?

Example: Write a program to read a set of integers and show them one by one

```
void process( ) {
  int num;
  A1 cin >> num;
  std::vector<int> s;
  for ( int i = 0; i < num; ++i ) {
        int input;
        A2
        cin >> input;
        s.push_back( input );
  }
  for ( int i = 0; i < s.size( ); ++i ) {
        A3
  }
}
```

```
// headers?
#include <iostream>
#include <vector>
using namespace std;
int main() {
  process( );
  return 0;
}
```

What are the problems of this program?

Display messages to guide the user about what to do next.

How to use std::vector

vector <data_type> obj; // create an object as a vector. The element type is data_type.

How to use std::vector

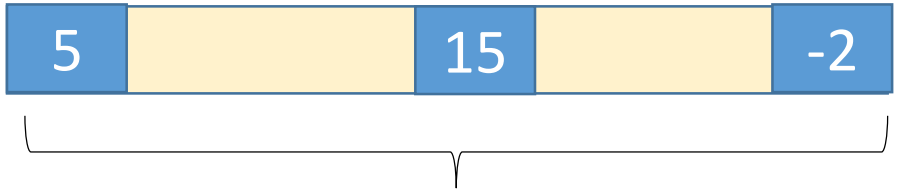
vector <data_type> obj; // create an object as a vector. The element type is data_type.

```
vector < float > s;          // s is empty
s.push_back( 10 );           // store the element to the back
s.push_back( 3 );            // store the element to the back
s.push_back( 7 );            // store the element to the back
```

	10	3	7
index	0	1	2

s[0] is 10
s[1] is 3
s[2] is 7

```
vector < int > t(20);        // store up to 20 elements
t[0] = 5;
t[10] = 15;
t[19] = -2;
```



How to use std::vector

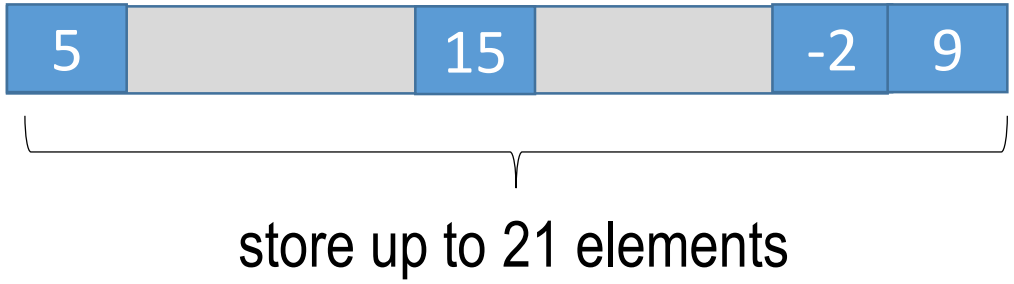
vector <data_type> obj; // create an object as a vector. The element type is data_type.

```
vector < float > s;          // s is empty
s.push_back( 10 );           // store the element to the back
s.push_back( 3 );            // store the element to the back
s.push_back( 7 );            // store the element to the back
```

	10	3	7
index	0	1	2

s[0] is 10
s[1] is 3
s[2] is 7

```
vector < int > t(20);        // store up to 20 elements
t[0] = 5;
t[10] = 15;
t[19] = -2;
t.push_back( 9 );
```



t[19] is ?
t[20] is ?
t[21] is ?

How to use std::vector

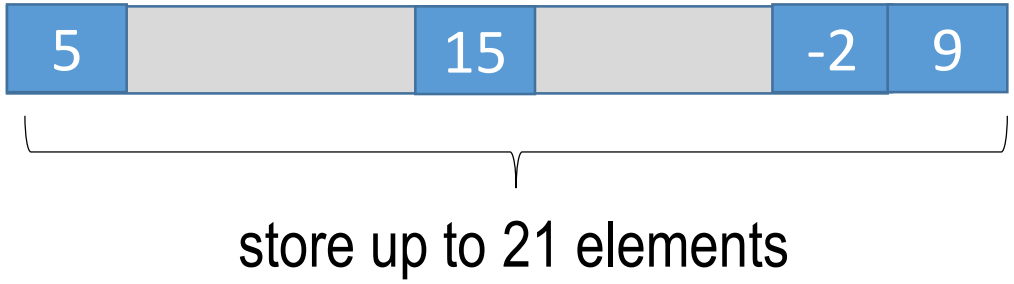
vector <data_type> obj; // create an object as a vector. The element type is data_type.

```
vector < float > s;          // s is empty
s.push_back( 10 );           // store the element to the back
s.push_back( 3 );            // store the element to the back
s.push_back( 7 );            // store the element to the back
```

	10	3	7
index	0	1	2

s[0] is 10
s[1] is 3
s[2] is 7

```
vector < int > t(20);        // store up to 20 elements
t[0] = 5;
t[10] = 15;
t[19] = -2;
t.push_back( 9 );
```



t[19] is ?
t[20] is ?
t[21] is ?

-2
9
error

Common errors in using std::vector

- Use it before it is not initialized
  - **vector<int> s;**
    s[0] = 11; // s is empty. Cannot use s[0].

- Out of bound
  - **vector<int> s(10);**
    s[10] = 21; // valid only for s[0], s[1], …, s[9]. Index 10 is out of range

- Have set its size and then store elements at the back. Miss to set the beginning elements
  - **vector<int> s(10);** // 10 elelemts
    s.push_back(21); // s[10] = 21. But s[0], s[1], …., s[9] are not set yet.
    s.push_back(32); // s[10] = 21. s[11] = 32. But s[0], s[1], …., s[9] are not set yet.

Variables

A variable represents a

A1 that may be

A2 in the program.

Variables

A variable represents a value that may be changed in the program.

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
cout << area;
```

Variables

A variable represents a value that may be changed in the program.

```
// Compute the first area
radius = 1.0;
area = radius * radius * 3.14159;
cout << area;

// Compute the second area
radius = 2.0;
area = radius * radius * 3.14159;
cout << area;
```

Declaring Variables

```
int x;            // Declare x to be an
                  // integer variable;
double radius;    // Declare radius to
                  // be a double variable;
char a;           // Declare a to be a
                  // character variable;
```

Assignment Statements

```
int x;
double radius;
char a;

x = 3;              // Assign 3 to x;
radius = 2.0;       // Assign 2.0 to radius;
a = 'A';            // Assign 'A' to a;
```

Declaration and Initialization

- `int x = 3;`
- `double d = 2.3;`
- `char a = ‘W’;`
- `string str = “hello world!”;`

Named Constants

```
const datatype CONSTANTNAME = VALUE;

const double PI = 3.14159;
const int SIZE = 3;
```

Make the program readable

```
a = d*d*c;
```

```
area = radius*radius*PI;
```

Named Constants

A named constants represent a permanent value.

```
const datatype CONSTANTNAME = VALUE;
```

```
const double PI = 3.14159;
const int SIZE = 3;
```

Make the program readable

```
a = d*d*c;
```

```
area = radius*radius*PI;
```

Numerical Data Types

short
unsigned short
int
unsigned int
long long
float
double
long double

What is the size of each data type?
i.e., the number of bytes to represent the data type

Numerical Data Types

short
unsigned short
int
unsigned int
long long
float
double
long double

What is the size of each data type?

```
cout << sizeof ( type) << endl;
```

e.g.,

```
cout << sizeof(long long) << endl;

cout << sizeof(int) << "\t" << sizeof(short) << endl;
```

measured in bytes

Numeric Literals: Constant values

A character-string: characters are selected from

the digits 0 through 9,

a sign character (+ or -),

and the decimal point.

For example, 36, 10001, and 5.3 are literals in the following statements:

```
int i = 36;
long k = 10001;
double d = 5.3;
```

octal and hex literals

By default, an integer literal is a decimal number.

```
cout << 10 << end;                    // decimal

cout << 0xFFFF << " " << 010;
```

0xFFFF: hexadecimal

010 : octal

Double (64-bit) vs. Float (32-bit)

The double type values are more accurate than the float type values. For example,

```
cout.precision(16);
cout << 1 /3.0f << endl;
```

0.3333333432674408 (correct up to 7 digit places)

```
cout << 1.0 / 3.0 << endl;
```

0.3333333333333333 (correct up to 16 digit places)

Numeric Operators

Name	Meaning	Example	Result
+	Addition	2 + 3	5
-	Subtraction	2 – 3	-1
*	Multiplication	2*3	6
/	Division	2/3	0
		2.0/3	0.666666
%	Remainder	6 % 5	1
		15%6	3 // 2*6 + 3
		(-15)%6	? // -3*6 + 3

Integer Division vs Floating Point Division

5 / 2 yields an integer 2. Integer division

5.0 / 2 yields a double value 2.5. Floating point division

Remainder Operator

Remainder is very useful in programming.
An even number % 2 is always 0 and an odd number % 2 is always 1.

Remainder Operator

Remainder is very useful in programming.
An even number % 2 is always 0 and an odd number % 2 is always 1.

(number%2) == 0

If it is true, number is even.

Remainder Operator

Remainder is very useful in programming.
An even number % 2 is always 0 and an odd number % 2 is always 1.

(number%2) == 0

If it is true, number is even.

```
If ( (number%2)  == 0 )
   cout << “Even number” << endl;
else cout << “Odd number” << endl;
```

Remainder Operator

Today is Saturday.
What is the day after one day?
What is the day after two days?
What is the day after 10 days later?

Remainder Operator

Today is Saturday.	0	Today is Saturday.
What is the day after one day?	0, 1	Sunday
What is the day after two days?	0, 1, 2	Monday
What is the day after 10 days later?	**0, 1, 2, 3, 4, 5, 6**, **7**, 8, 9, 10	
	0, 1, 2, 3, 4, 5, 6, **0**, 1, 2, 3	

Sat, Sun, Mon, Tue

Remainder Operator

Today is Saturday.
What is the day after one day?
What is the day after two days?
What is the day after 10 days later?

0	1	2	3	4	5	6
Sun	Mon	Tue	Wed	Thu	Fri	Sat

Remainder Operator

Today is Saturday.

What is the day after one day?	(6+1)%7
What is the day after two days?	(6+2)%7
What is the day after 10 days later?	(6+10)%7

0	1	2	3	4	5	6
Sun	Mon	Tue	Wed	Thu	Fri	Sat

Remainder Operator

Today is Saturday.

What is the day after one day?	(6+1)%7	0	// 7 % 7
What is the day after two days?	(6+2)%7	1	// 8 % 7
What is the day after 10 days later?	(6+10)%7	2	// 16 % 7

0	1	2	3	4	5	6
Sun	Mon	Tue	Wed	Thu	Fri	Sat

The period of a cycle: 7 units.

(1+7)%7 = (1+7*2)%7 = (1+7*n)%7 = 1;

Augmented Assignment Operators

Operator	Example	Equivalent
`+=`	`i += 2`	`i = i + 2`
`-=`	`f -= 2.0`	`f = f - 2.0`
`*=`	`i *= 2`	`i = i * 2`
`/=`	`i /= 2`	`i = i / 2`
`%=`	`i %= 2`	`i = i % 2`

Shorthand operators

Increment and Decrement Operators

Operator	Name	Description
++v	preincrement	(++v) increments v by 1 and evaluates to the new value in v after the increment.
v++	postincrement	(v++) evaluates to the original value in v and increments v by 1.
--v	predecrement	(--v) decrements v by 1 and evaluates to the new value in v after the decrement.
v--	postdecrement	(v--) evaluates to the original value in v and decrements v by 1.

They modify the variable but also return a value.

Increment and Decrement Operators

Operator	Name	Description
++v	preincrement	(++v) increments v by 1 and evaluates to the new value in v after the increment.
v++	postincrement	(v++) evaluates to the original value in v and increments v by 1.
--v	predecrement	(--v) decrements v by 1 and evaluates to the new value in v after the decrement.
v--	postdecrement	(v--) evaluates to the original value in v and decrements v by 1.

How do they work?

Increment and Decrement Operators

Operator	Name	Description
++v	preincrement	(++v) increments v by 1 and evaluates to the new value in v after the increment.

How do they work?

Increment and Decrement Operators

Operator	Name	Description
++v	preincrement	(++v) increments v by 1 and evaluates to the new value in v after the increment.

How do they work?

Process from A1
When the variable is A2, use the A3
When the operator is encountered, apply it to the A4

Increment and Decrement Operators

```
int i = 11;
int newValue = 5 *  i++;

int a = 11;
int newA = 5* ++a;
```

Process from left to right.
When the variable is encountered, use the value of the variable.
When the operator is encountered, apply it to the variable.

Increment and Decrement Operators

```
int i = 11;
int newValue = 5 *  i++;
```

```
int i = 11;
int newValue = 5 * i;
i = i + 1;
```

```
int a = 11;
int newA = 5* ++a;
```

```
int a = 11;
a = a + 1;
int newA = 5*a;
```

Process from left to right.
When the variable is encountered, use the value of the variable.
When the operator is encountered, apply it to the variable.

Increment and Decrement Operators

```
int i = 11;
int newValue = 5 *  i++;
```

```
int i = 11;
int newValue = 5 * i;
i = i + 1;
```

newValue is 55
i is 12

```
int a = 11;
int newA = 5* ++a;
```

```
int a = 11;
a = a + 1;
int newA = 5*a;
```

newA is 60
a is 12

Process from left to right.
When the variable is encountered, use the value of the variable.
When the operator is encountered, apply it to the variable.

Numeric Type Conversion

```
short i = 101;
long k = i * 7 + 4;
double d = i * 3.2 + k / 2;
```

Type Casting

```
Implicit casting
  double d = 7; (type widening)

Explicit casting
  int i = static_cast<int>(7.0); (type narrowing)
  int i = (int)7.9; (Fraction part is truncated)
```

static_cast: it is a compile time cast.

Type Casting

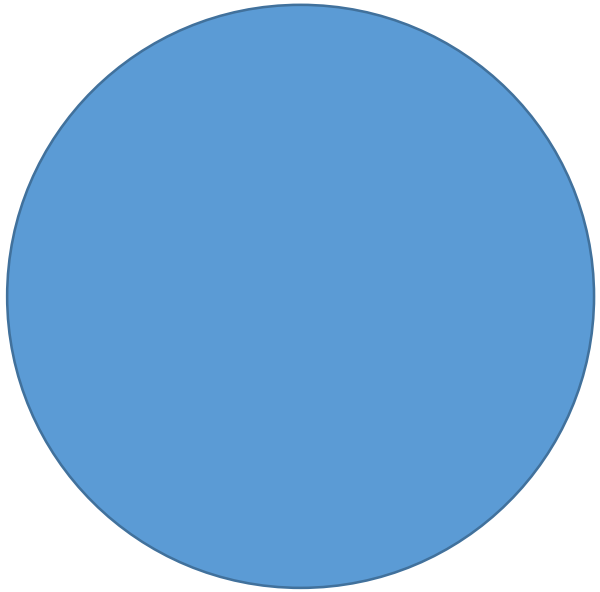
The variable which is being cast does not change.

```
double d = 7.5;
int i = static_cast<int>(d);  // d is not changed
```

Supplemental Material

Computing the Area of a Circle

- area = radius*radius*pi



Use white space to make the program readable

```
#include <iostream>
using namespace std;
int main() {
  double radius;
  double area;
  // Step 1: Read in radius
  radius = 20;
  // Step 2: Compute area
  area = radius * radius * 3.14159;
  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```

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

int main() {
  double radius;
  double area;

  // Step 1: Read in radius
  radius = 20;

  // Step 2: Compute area
  area = radius * radius * 3.14159;

  // Step 3: Display the area
  cout << "The area is ";
  cout << area << std::endl;
}
```

Identifiers

- A sequence of characters that consists of letters, digits, and underscores (_).

  e.g., this_is_an_identifier_1453_, _variable

- Must start with a letter or an underscore.
- Cannot start with a digit.

  e.g., 1623_is_not_an_identifier

- An identifier cannot be a reserved word.

  e.g., class is a keyword

- Use identifiers of 1311 characters or fewer.

Regular expression: [a-z, A-Z, _]{ a-z, A-Z, _, 0-9 }*

floating-point numbers

35.3242 x 10^0

0.353242 x 10^2

- radix point: separate the integer part of a number from its fractional part

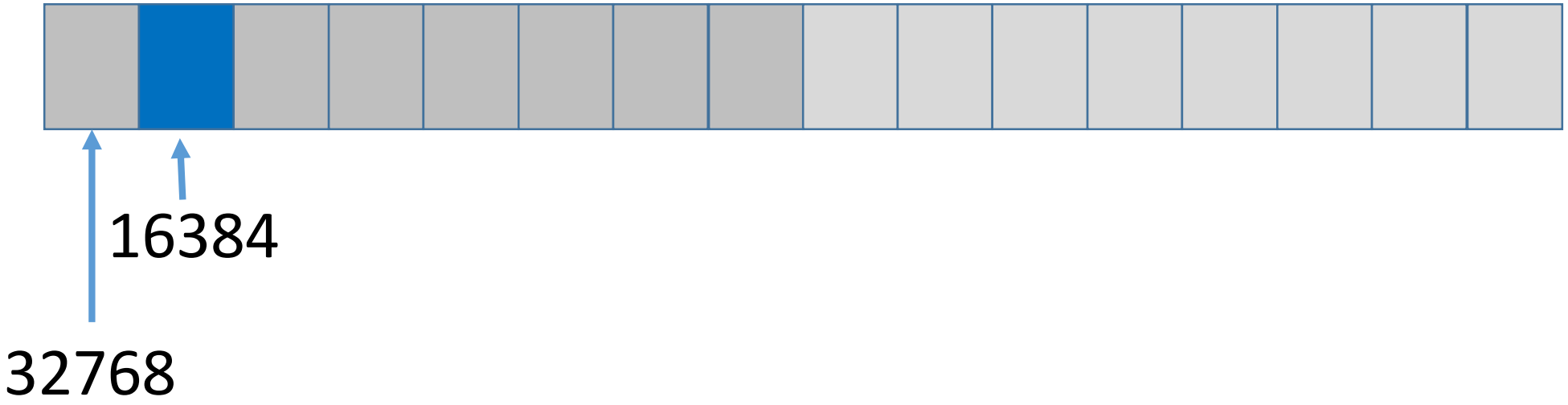
Exponent Operations

```
cout << pow(2.0, 3) << endl; // Display 8.0
cout << pow(4.0, 0.5) << endl; // Display 2.0
cout << pow(2.5, 2) << endl; // Display 6.25
cout << pow(2.5, -2) << endl; // Display 0.16
```

Overflow

When a variable is assigned a value that is too large to be stored, it causes *overflow*.

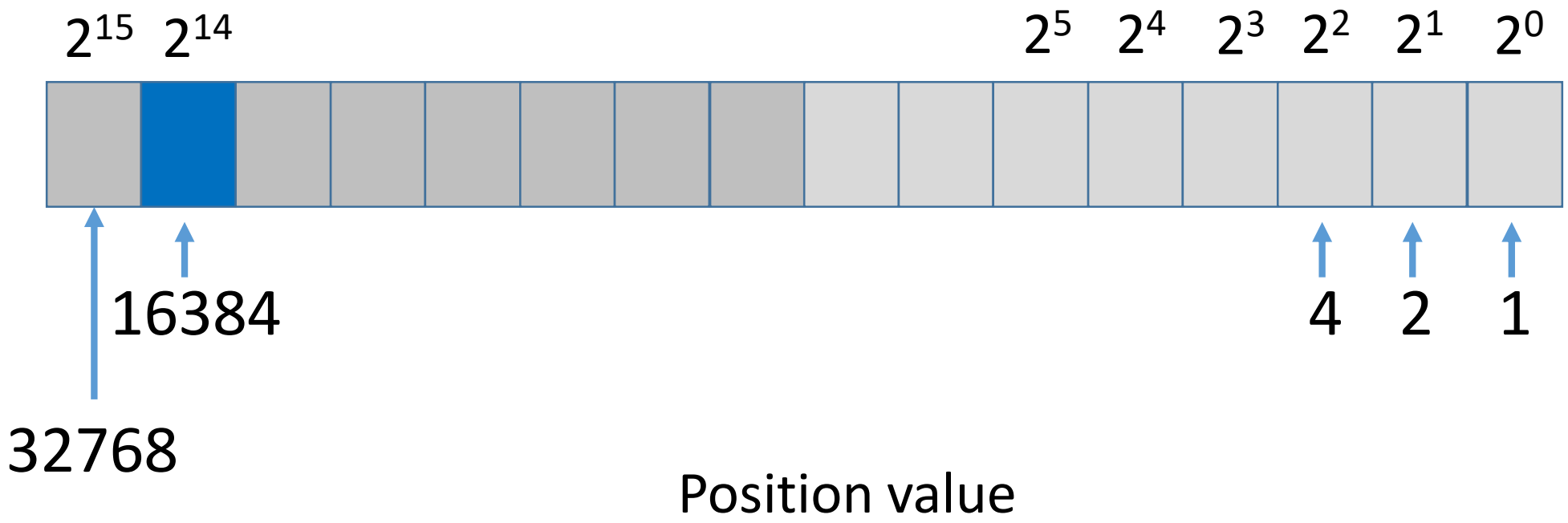
short value = 32767 + 1; // this is a signed integer



Overflow

When a variable is assigned a value that is too large to be stored, it causes *overflow*.

short value = 32767 + 1; // this is a signed integer



Arithmetic Expressions

$$\frac{2+5x}{y+4z} - \frac{12+5x}{8y-4w}$$

is translated to

(2+5*x)/(y+4*z) – (12+5*x)/(8*y - 4*w)

Increment and Decrement Operators

```
int k = ++i + i;
```

//avoid doing this

// how do we interpret the instruction?

Increment and Decrement Operators

```
int k = ++h( ) + h( );

//avoid doing this
// how do we interpret the instruction?
```

Increment and Decrement Operators

Potential problem

```
int x = 2;
int a = 0;

x = ++x - --x;
x = ++x - --x + ++a - a--;
```

//avoid doing this

//why?