CS2313 Computer Programming

LT3 – Language Syntax, Variable, Data types and Basic I/O-Part II

香港城市大學
City University of Hong Kong
專業 創新 胸懷全球
Professional·Creative
For The World

Overflow

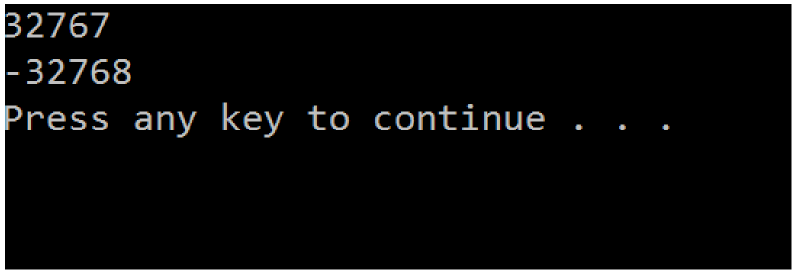
When a variable is assigned a value that is too large to be stored, it causes overflow.

For example, executing the following statement causes overflow, because the largest value that can be stored in a variable of the short type is 32767. 32768 is too large.

```
short value = 32767 + 1;
```

Overflow

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

int main()
{
    short value0 = 32767;
    short value1 = 32768;

    cout << value0 <<endl;
    cout << value1 <<endl;

    return 1;
}
```

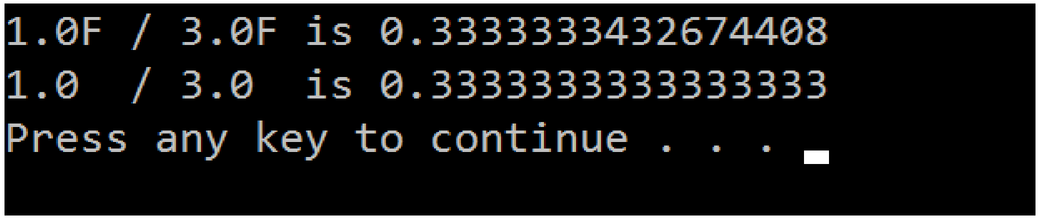


```
32767
-32768
Press any key to continue . . .
```

double vs. float

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

```
cout << "1.0F / 3.0F is " << setprecision(16)<< 1.0F / 3.0F << endl;
cout << "1.0  / 3.0  is " << setprecision(16)<< 1.0  / 3.0  << endl;
```



```
1.0F / 3.0F is 0.3333333432674408
1.0  / 3.0  is 0.3333333333333333
Press any key to continue . . .
```

A suffix can be appended to a floating constant to specify its type; the default is double which is the working floating type in C++.

why called floating-point?

The float and double types are used to represent numbers with a decimal point. Why are they called floating-point numbers? These numbers are stored into scientific notation. When a number such as 505.34 is converted into scientific notation such as 5.0534e+2, its decimal point is moved (i.e., floated) to a new position.

The World is Not Just Integers

Type Conversion

Arithmetic **conversions** occur when necessary as the operands of a binary operator are evaluated.

```
short  i = 100;
long   k = i * 3 + 4;
double d = i * 3.1 + k / 2;
```

Type Conversion

- Implicit type conversion

  double d = 3; (type widening)

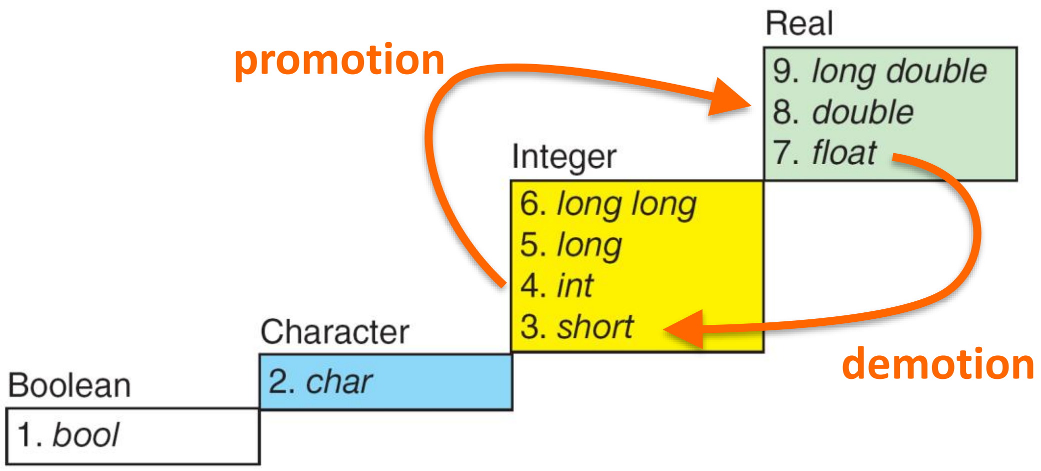
- Explicit type conversion

  int i = static_cast<int>(3.0); (type narrowing)

  int i = (int)3.9; (Fraction part is truncated)

Type Conversion

- Implicit type conversion (casting)
  - binary expressions (e.g. `x + y`): lower-ranked operand is promoted to higher-ranked operand.
  - assignment (e.g. `x = y`): right operand is promoted/demoted to match the variable type on the left.
- Explicit type conversion
  - example: `j = (double) i;`
- Demoted values might change or become invalid.



Type Conversion (Note)

- Type conversion may not change the variable being cast.
  - For example, d is not changed after casting in the following code

```
double d = 4.5;
int         i = static_cast<int>(d);  // d is not changed
```

- For user defined data types, the complier does not support automatic type conversions.

- We must define the conversion routines by ourselves

Constants

- Like variable, constant store data for program access but its value will not be changed after declaration.

```
const float pi = 3.14159;
```

- Constants can be numeric-, character- or string-based.
  - e.g., `13, 7.11, '\n', "Tuesday".`

- Numbers are represented in **decimal** system but **octal** (preceded by 0) or **hexadecimal** integers (preceded by 0x) can be represented, e.g., the following number are equal to a decimal `26.`

```
032        /* an octal integer */
0x1a       /* an hexadecimal integer  */
```

- Character constant is enclosed by single quotation marks.
  - e.g., `'a', '\n', '\t'.`

Numeric Literals

A **literal** is a constant value that appears directly in a program. For example, 34, 1000000, and 5.0 are literals in the following statements:

```
int i = 34;
long k = 1000000;
double d = 5.0;
```

octal and hex literals

By default, an integer literal is a decimal number. To denote an octal integer literal, use a leading 0 (zero), and to denote a hexadecimal integer literal, use a leading 0x or 0X (zero x). For example, the following code displays the decimal value 65535 for hexadecimal number FFFF and decimal value 8 for octal number 10.

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

Declaration – Constant

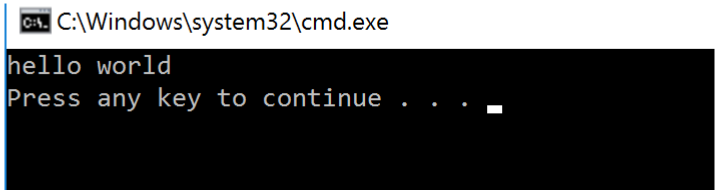
- Format:
  - const Data_type variable/constant identifier = value;
- Examples:

```
const float pi = 3.14159;
const int maxValue = 500;
const char initial = ‘D’;
const char student_name[] = ”John Chan”;
```

Constants

- String constants are delimited by double quotes (").
- String is treated as an *array* of characters in C++
  - C++ call it cstring.
  - Another type of string is `String` object which will be covered in the future lecture.

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

int main()
{
    const string a = "hello world";
    cout << a <<endl;

    return 1;
}
```



C:\Windows\system32\cmd.exe

```
hello world
Press any key to continue . . .
```

Operators and Punctuators

Operators and Punctuators

- Punctuators and operators are used to separate language elements, e.g.,

```
int a, b = 4 , c = 4;

a = b + c;
```

- Some operators:

```
+, -, *, /, %, ++, --, >>, <<.
```

- Some symbols have meaning that depends on context, e.g.,

```
printf("%d", 40%7);
```

Numeric Operators

Name	Meaning	Example	Result
`+`	Addition	`34 + 1`	`35`
`-`	Subtraction	`34.0 – 0.1`	`33.9`
`*`	Multiplication	`300 * 30`	`9000`
`/`	Division	`1.0 / 2.0`	`0.5`
`%`	Remainder	`20 % 3`	`2`

Integer Division

+, -, *, /, and %

5 / 2 yields an integer 2.

5.0 / 2 yields a double value 2.5

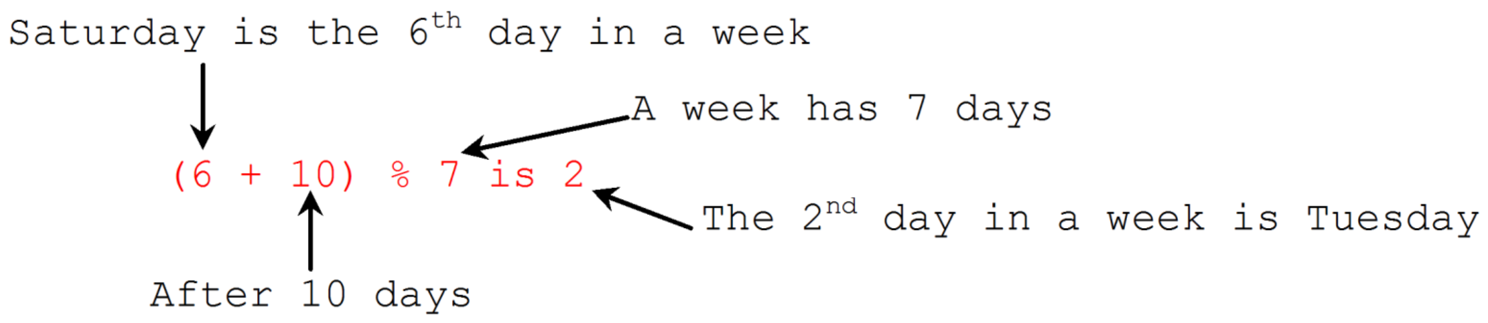
5 % 2 yields 1 (the remainder of the division)

Remainder Operator

Remainder is very useful in programming.

For example, an even number % 2 is always 0 and an odd number % 2 is always 1.

Suppose today is Saturday and you and your friends are going to meet in 10 days. What day is in 10 days?


Saturday is the 6th day in a week
A week has 7 days
(6 + 10) % 7 is 2
The 2nd day in a week is Tuesday
After 10 days

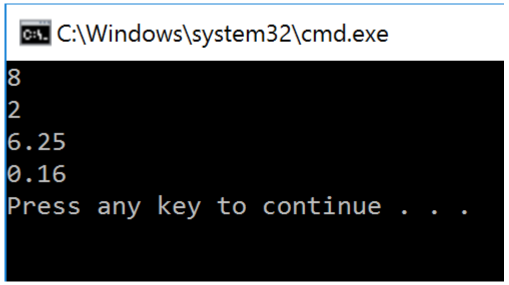
Bitwise Operator

Assuming variable A holds 60 (00111100) and variable B holds 13 (00001101)

Operator	Description	Example
&	Binary AND operator copies a bit to the result if it exists in both operands.	A&B will give 12 which is 00001100
|	Binary OR operator copies a bit to the result if it exists in either operands.	A|B will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit (1) to the result if it exists in one operand but not both	A^B will give 49 which is 0011 0001
<<	Binary left shift operator, moved left by the number of bits specified by the right operand	A<<2 will give 240, which is 1111 0000
>>	Binary right shift operator, moved right by the number of bits specified by the right operand	A>>2 will give 15, which is 0000 1111

Exponent Operations

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

int main()
{

    cout << pow(2.0, 3)  << endl;
    cout << pow(4.0, 0.5)<< endl;
    cout << pow(2.5, 2)  << endl;
    cout << pow(2.5, -2) << endl;

    return 1;
}
```



```
8
2
6.25
0.16
Press any key to continue . . .
```

Increment & Decrement Operators

- Increment and decrement operators: `++` and `--`
  - `k++` and `++k` are equivalent to `k=k+1`.
  - `k--` and `--k` are equivalent to `k=k-1`.

- Post-increment and post-decrement: `k++` and `k--`
  - `k`'s value is altered **AFTER** the expression is evaluated.

```
int k=1, j;
j=k++;    /* result: j==1, k==2 */
```

- Pre-increment and pre-decrement: `++k` and `--k`
  - `k`'s value is altered **BEFORE** the expression is evaluated.

```
int k=1, j=0;
j=++k;    /* result: j==2, k==2 */
```

Precedence and Associativity of Operators

- An expression may have more than one operator and its precise meaning depends on the precedence and associativity of the involved operators.

- What is the value of variables a, b and c after the execution of the following statements

```
    int a, b = 2, c = 1;
    a = b+++c;
```

- Which of the following interpretation is right?

```
    a = (b++) + c; /* right */
or  a = b + (++c); /* wrong */
```

Precedence and Associativity of Operators

Precedence Decreases

Operator Precedence (high to low)	Associativity
::	None
. -> []	Left to right
() ++(postfix) --(postfix)	Left to right
+ -(unary) ++ (prefix) -- (prefix)	Right to left
* / %	Left to right
+ -	Left to right
= += -= *= /= etc.	Right to left

Precedence: order of evaluation for different operators.

Associativity: order of evaluation for operators with the same precedence.

Precedence and Associativity of Operators

Binary plus

Binary multiplication

Binary minus

```
int x;
x = 3 + 4 * 5 -6;

x = 3 + (4 * 5) -6;

x = (3 + (4 * 5)) -6;
```

Multiplication has higher precedence

Left to right

```
int a, b = 2, c = 1;
    a = b+++c;

    a = (b++)+c;
```

b's value is altered **AFTER** the expression is evaluated.

a=3; b=3; c=1;

```
a = b+++c;
```

Never write program like this!

Assignment Operators =

- Generic form

  variable = expression;

- An expression itself has a value, e.g.,

```
a = (b = 32) + (c = 23);
```

Examples on Assignment Statement

```
/* Invalid: left hand side must be a variable */
a + 10 = b;

/*assignment to constant is not allowed*/
2=c;

/* valid but not easy to understand */
int a, b, c;
a = (b = 2) + (c = 3);

/* avoid complex expressions*/
int a, b, c;
b = 2;
c = 3;
a = b + c;
```

Swapping the Values

- If we want to swap the content of two variables, a and b.
- What's the problem for the following program?

```
void main(){
  int a=3, b=4;
  a=b;
  b=a;
}
```

Swapping the Values

- We need to make use of a temporary variable.

```
c=b; /*save the old value of b*/
b=a; /*put the value of a into b*/
a=c; /*put the old value of b to a*/
```

Assignment Operators

- Generic form of efficient assignment operators

  variable op= expression;

  where *op* is operator; the meaning is

  variable = variable op (expression);

- Efficient assignment operators include:

```
+=     -=     *=     /=     %=     >>=
<<=    &=     ^=     |=
```

- Examples:

```
a += 5;      is same as      a = a + 5;
a -= 5;      is same as      a = a - 5;
a += b*c;    is same as      a = a + (b*c);
a *= b+c;    is same as      a = a * (b+c);
```

- `=` is an assignment operator that has nothing to do with mathematical equality (which is `==` in C++)

`a++` and `++a`

Two things happen:

(1) Compute the value of the expression `a++` or `++a`.

(2) Increment `a` by `1`.

`a++`

do (1) before (2).

Therefore, the value of `a++` is equal to old value of `a`.

`++a`

do (2) before (1).

Therefore, the value of `++a` is equal to the incremented value.

Example

```
int x=3;
cout << x;
cout << ++x;
cout << x;
cout << x++;
cout << x;
```

Output

	Old x	New x	Output
`int x=3;`	3	3	
`cout <<  x;`	3	3	3
`cout << ++x;`	3	4	4
`cout << x;`	4	4	4
`cout << x++;`	4	5	4
`cout << x;`	5	5	5

What Values Are Printed?

```
int a=0,i=0;
cout << "i= " << i << endl;

a=0;
i=1+(a++);
cout << "i= " << i << endl;
cout << "a= " << a << endl;


a=0;
i=1+(++a);
cout << "i= " << i << endl;
cout << "a= " << a << endl;
```

a++ and ++a

$$\texttt{i=1+}\underbrace{\texttt{(a++)}}_{0}\texttt{;}$$

Evaluates `a++` (value:0)

computes `a=a+1`

```
i=1+0
 =1
```

$$\texttt{i=1+}\underbrace{\texttt{(++a)}}_{1}\texttt{;}$$

computes `a=a+1`

evaluates `++a` (value:1)

```
i=1+1
 =2
```

Answer

```
int a,i;
i=(a=0);
cout << i= " << i << endl;

a=0;
i=1+(a++);
cout << "i= " << i << endl;
cout << "a= " << a << endl;


a=0;
i=1+(++a);
cout << "i= " << i << endl;
cout << "a= " << a << endl;
```

Output

```
i=0
i=1
a=1
i=2
a=1
```

Programming Styles

- **Programmers should write code that is:**
  - **understandable to other programmers as well.**
- Meaningful variable names.
- Which one is more meaningful:

```
tax = temp1*temp2;
tax = price*tax_rate;
```

- Meaningful Comments.
  - Write comments as you're writing the program.
- Indentation.

Use of Comments

- Top of the program:
  - Include information such as *the name of organization, programmer's name, date and purpose of program*.
- What is achieved by the function, the meaning of the arguments and the return value of the function.
- Short comments should occur to the right of the statements when the effect of the statement is not obvious and you want to illuminate what the program is doing.

Which one of the following is more meaningful?

```
tax = price * rate; /* sales tax formula */
tax = price * rate; /* multiply price by rate */
```

Common Errors

- Common Error 1: Undeclared/Uninitialized Variables and Unused Variables
- Common Error 2: Integer Overflow
- Common Error 3: Round-off Errors
- Common Error 4: Unintended Integer Division
- Common Error 5: Forgetting Header Files

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount 1156

remainingAmount initialized

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	1156
numberOfOneDollars	11

numberOfOneDollars assigned

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	56
numberOfOneDollars	11

remainingAmount updated

Trace ComputeChange

Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

remainingAmount	56
numberOfOneDollars	11
numberOfOneQuarters	2

numberOfOneQuarters assigned

Trace ComputeChange

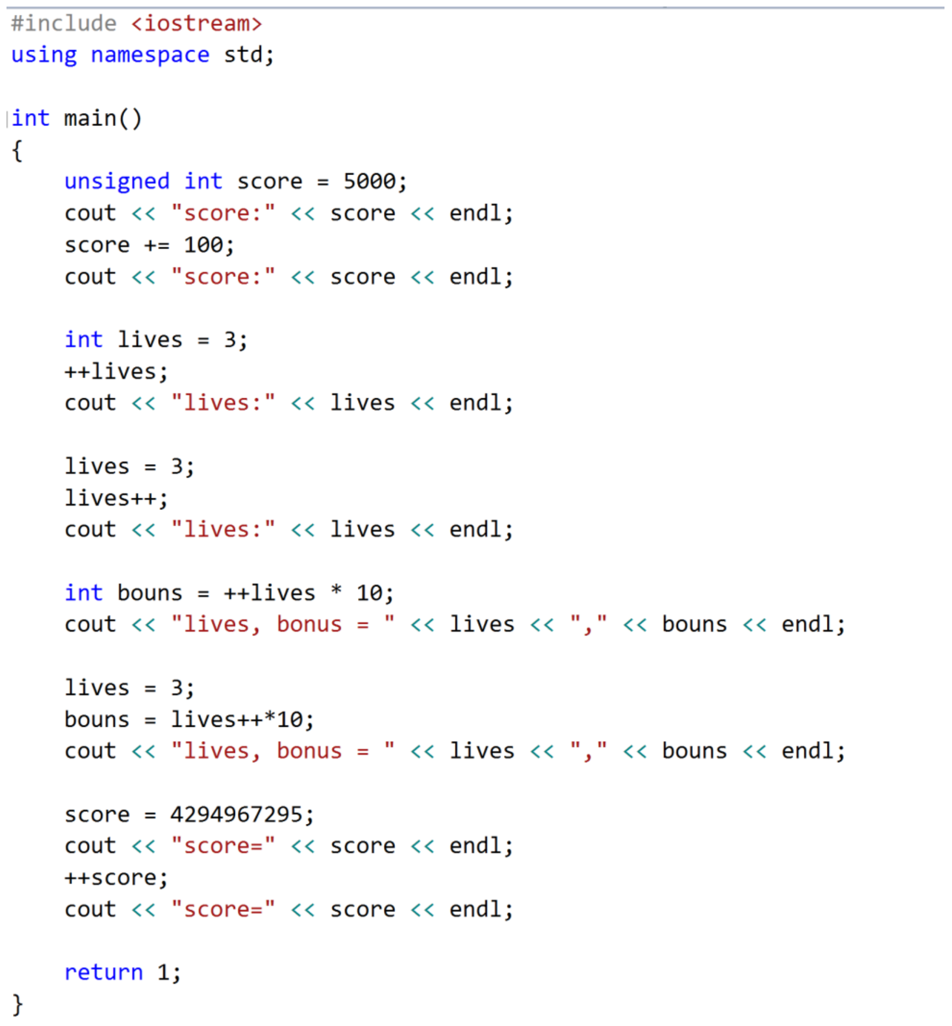
Suppose amount is 11.56

```
int remainingAmount = (int)(amount * 100);

// Find the number of one dollars
int numberOfOneDollars = remainingAmount / 100;
remainingAmount = remainingAmount % 100;

// Find the number of quarters in the remaining amount
int numberOfQuarters = remainingAmount / 25;
remainingAmount = remainingAmount % 25;

// Find the number of dimes in the remaining amount
int numberOfDimes = remainingAmount / 10;
remainingAmount = remainingAmount % 10;

// Find the number of nickels in the remaining amount
int numberOfNickels = remainingAmount / 5;
remainingAmount = remainingAmount % 5;

// Find the number of pennies in the remaining amount
int numberOfPennies = remainingAmount;
```

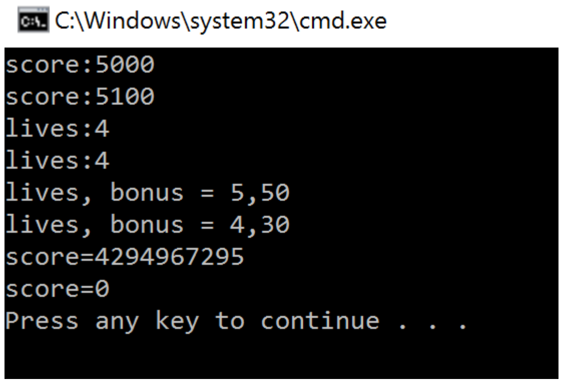
remainingAmount	6
numberOfOneDollars	11
numberOfQuarters	2

remainingAmount updated

Gaming Statistics



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

int main()
{
    unsigned int score = 5000;
    cout << "score:" << score << endl;
    score += 100;
    cout << "score:" << score << endl;

    int lives = 3;
    ++lives;
    cout << "lives:" << lives << endl;

    lives = 3;
    lives++;
    cout << "lives:" << lives << endl;

    int bouns = ++lives * 10;
    cout << "lives, bonus = " << lives << "," << bouns << endl;

    lives = 3;
    bouns = lives++*10;
    cout << "lives, bonus = " << lives << "," << bouns << endl;

    score = 4294967295;
    cout << "score=" << score << endl;
    ++score;
    cout << "score=" << score << endl;

    return 1;
}
```



C:\Windows\system32\cmd.exe

```
score:5000
score:5100
lives:4
lives:4
lives, bonus = 5,50
lives, bonus = 4,30
score=4294967295
score=0
Press any key to continue . . .
```

Gaming Statistics

```
unsigned int score = 5000;
cout << "score:" << score << endl;
score += 100;
cout << "score:" << score << endl;
```

score = 5000;
score = 5100

```
int lives = 3;
++lives;
cout << "lives:" << lives << endl;

lives = 3;
lives++;
cout << "lives:" << lives << endl;
```

lives = 4;
lives = 4;

Gaming Statistics

```
int bouns = ++lives * 10;
cout << "lives, bonus = " << lives << "," << bouns << endl;

lives = 3;
bouns = lives++*10;
cout << "lives, bonus = " << lives << "," << bouns << endl;
```

lives = 5; bonus = 50;
lives = 4; bonus = 30;

```
score = 4294967295;
cout << "score=" << score << endl;
++score;
cout << "score=" << score << endl;
```

score = 4294967295
score = 0; (overflow!)

Summary

- Type conversion
  - Implicit type conversion
  - Explicit type conversion
- Constants
  - String
  - Numeric
- Operators and Punctuators
  - Symbols in different contexts have different meanings
  - Swapping the values
  - a++ and ++a
- Programming styles
  - Comments
  - Meaningful variable names.