C++ & Python Program Design

-- Literals, Consts &Variables

Junjie Cao @ DLUT

Summer 2022

https://github.com/jjcao-school/c

Number Data Types

Number Data Types

- Python:
  - integers,
  - floating-point numbers,
  - complex numbers.

```
>>> type(1)
<class 'int'>
>>> type(1.0)
<class float'>

>>> 1e-4
0.0001
```

- C++:
  - int for integer,
  - float for floating point,
  - double for double precision floating point.

C++ Number Data Types

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

// Function that perfoms
various math operations
int main(){

    cout << (2+3*4) << endl;
    cout << (2+3)*4 << endl;
    cout << pow(2, 10) << endl;
    cout << float(6)/3 << endl;
    cout << float(7)/3 << endl;
    cout << 7/3 << endl; //In C++ this is
integer division
    cout << 7%3 << endl;
    cout << float(3)/6 << endl;
    cout << 3/6 << endl;
    cout << 3%6 << endl;
    cout << pow(2, 100) << endl;

    return 0;
}
```

```
print(2+3*4)
print((2+3)*4)
print(2**10)
print(6/3)
print(7/3)
print(7//3)
print(7%3)
print(3/6)
print(3//6)
print(3%6)
print(2**100)
```

Boolean Data

```cpp
// function that demonstrates
logical operators
int main() {
cout << true << endl; //1
cout << false << endl; //0
cout << (true || false) << endl; //1
cout << (true && false) << endl; //0
return 0;
}
```

- C++: 0 == false, !0 == true

```python
print(True) #True
print(False) #False
print(True or False) #True
print(True and False) #False
```

Relational and Logical Operators 操作符

```
//demonstrates relational operators.
cout << (5 == 10) << endl; //0
cout << (10 > 5) << endl; //1
cout << ((5 >= 1) && (5 <= 10)) << endl; //1
```

```
print(5 == 10)#False
print(10 > 5)#True
print((5 >= 1) and (5 <= 10))#True
```

- C++: 0 == false, !0 == true

Literal Constants字面值常量

- C++ Example: 看一眼就行，不要记

20 //decimal int
// Example: int i = 20;
024 //octal
0x14 //hexadecimal
3.14F // decimal float
3.14E0f // decimal float, scientific notation
1e-3F
'O'
"Hello word"
L"a wide string literal"

Escape Sequences for Nonprintable Characters:
\n //newline
// Example: cout << "hi\n";
\t //horizontal tab
\v //vertical tab
\b //backspace

```
// ok: A \ before a newline ignores the line
break
std::cou\
t << "Hi" << st\
d::endl;
```

```
std::cout << "Hi" << std::endl;
```

Variable 变量

Create a Variable 变量

```
phrase = "Hello, world"
print(phrase)
```

```
std::string phase = "Hello world";
std::cout << phase << std::endl;
```

- **variables** are **names** that can be assigned a value and used to reference that value throughout your code.

- =: The **Assignment** Operator. Try this & compare the error info with :

```
#phrase = "Hello, world"
print(phrase)
```

```
print(Hello, world)
```

Variables变量

- Do you find the diff of the following code?

```
int theSum = 4;
cout << theSum << endl;
theSum = theSum + 1;
cout << theSum << endl;
```

```
theSum = 4
print(theSum)
theSum = theSum + 1
print(theSum)
```

- C++: **static typing,** have to declare it be it is used. Python **dynamic tying** (自己构造几行code试试？).
  - **Declaration**: specify data type of each variable.
  - Static: C++ variables cannot change type.

```
bool theBool; //try to comment this line
theBool = true; //try to comment this line
cout << theBool << endl;
```

Variables - Primitive Built-in Types

Type	Meaning	Minimum Size
bool	boolean	NA
char	character	8 bits
wchar_t	wide character	16 bits
short	short integer	16 bits
int	integer	16 bits
long	long integer	32 bits
float	single-precision floating-point	6 significant digits
double	double-precision floating-point	10 significant digits
long double	extended-precision floating-point	10 significant digits

- Size? Bit比特，bitcoin比特币

type() vs. typeid() & sizeof()

```
int iSum = 4; float fSum = 4; double dSum = 4;
cout << typeid(iSum).name() << ": " << sizeof(iSum) << endl;
cout << typeid(fSum).name() << ": " << sizeof(fSum) << endl;
cout << typeid(dSum).name() << ": " << sizeof(dSum) << endl;
```

size of i: 4

f: 4

d: 8

```
iSum = 4
fSum = 4.0
print("type of iSum is {} and that of fSum is
{}".format(type(iSum), type(fSum)))
```

```
type of iSum is <class 'int'> and that of fSum is <class
'float'>
```

- In Python, *everything* is an object.

Beware of assigning an out-of-range value when processing image

- char, 8bits, -128-127
- unsigned char, 8bits, 0-255

- int, 16bits, 0-65535
- unsigned int, 16bits, -32768-32767

```
unsigned char a(-1); //?
char b(128);//?
```

Variable names are case-sensitive

```
>>> phrase = "Hello, world"
>>> print(Phrase)

Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'Phrase' is not defined
```

Name of a Variable

- is an identifier token:
  - may contain numbers, letters, and underscores(_)
  - may **not start with a number**
  - case-sensitive
    - int x, X;
  - Not keyword

Which, if any, of the following names are invalid?
Correct each identified invalid name

(a)	int double = 3.14159;	Keyword
(b)	char _;	Ok
(c)	bool catch-22;	-
(d)	char 1_or_2 ='1';	Start with a number
(e)	float Float = 3.14f;	OK

Naming

- should be **descriptive**
- **eschew abbreviation避免缩写**
- Variable Names
  - variables should be **nouns**
  - all lowercase, with underscores between words
    - my_exciting_local_variable

- Constant Names
  - Use a k followed by mixed case
    - kDaysInAWeek

Scope of a Name in C++

- Delimited by curly braces {}

```
int i = 42;
int j = 0;
{
int i = 100;
j = i;
cout << j << endl;
}
cout << i << endl;
cout << j << endl;
```

- Best practice
  - Do **not use global** variable
  - define an object **near** the point at which the object is **first used**.
- Global scope vs local scope
- Class scope, Namespace scope

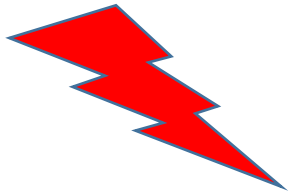
Python等讲到function再说。

Typedef names

1. allow a single type to be used for more than one purpose while **making the purpose clear** each time the type is used
   - typedef double wages; // wages is a synonym for double
   - typedef int exam_score; // exam_score is a synonym for int
   - typedef wages salary; // indirect synonym for double
   - wages hourly, weekly; // double hourly, weekly;
   - exam_score test_result; // int test_result;

2. streamline complex type definitions, making them easier to understand
   - typedef std::vector<Point<double,3>> Points;

Operator 操作符 & data types



- An operation can only be **performed** on **compatible types**
  - 34 + 3; //ok
  - 3%5.0; // compile error
    - you can't take the remainder of an integer & a floating-point number

- An operator also normally **produces a value** of the **same type** as its operands;
  - cout << 1/4;//0
    - because with two integer operands, / truncates the result to an integer.
  - cout << 1/4.0;//0.25

- Python
  - print(%5.0) # 3
  - print(1/4) #0.25

Type Conversions

```
1 int x = (int)5.0; // float should be explicitly "cast" to int
2 short s = 3;
3 long l = s; // does not need explicit cast, but
4             // long l = (long)s is also valid
5 float y = s + 3.4; // compiler implicitly converts s
6                    // to float for addition
```

- long a = sqrt(10);

1>… error C2668: 'sqrt' : ambiguous call to overloaded function

1> c:\program files\microsoft visual studio 10.0\vc\include\math.h(589): could be 'long double sqrt(long double)'

1> c:\program files\microsoft visual studio 10.0\vc\include\math.h(541): or 'float sqrt(float)'

1> c:\program files\microsoft visual studio 10.0\vc\include\math.h(127): or 'double sqrt(double)'

1> while trying to match the argument list '(int)‘

- long a = (float) sqrt(float (10));
- long a = (float) sqrt(10.0f);

Const 常量

A constant is an expressions 表达式 with a **fixed** value:

- Literals: particular values in source code, such as 2.14, ‘\n’
- Defined constants (#define)
- Delcared constants (const)

Nameless constants

Named constants

```cpp
#include <iostream>

using namespace std;

#define PI 3.14159
#define NEWLINE '\n'

int main()
{
    double r = 5.0;
    double circle;

    circle = 2 * PI * r; // circle = 2 * 3.14159 * r;
    cout << circle << NEWLINE; // cout << circle << '\n';

    const int pathwidth = 100;
}   // pathwidth = 2; this will cause a compiler error!
    const char tabulator = '\t';
    cout << "tabulator =" << tabulator << '\n';
```

#define -- In general, macros宏 should *not* be used.

```
【 以下文字转载自 CPlusPlus 讨论区 】
发信人: forcey (爱到无可救药), 信区: CPlusPlus
标  题: Re: 刚发现在vs studio里汉字也可以当变量名
发信站: 水木社区 (Fri May 25 05:53:50 2012), 站内

#define 趁还 while
#define 那个啥 int
#define 总的来说 main
#define 买 cin
#define 卖 cout
#define 进 >>
#define 出 <<
#define 拜拜了 return
#define 去掉 -=
#define 等于 =
#define 屁 100e4
#define 我说 (
#define 是吧 )
#define 啊 a
#define 那么就 {
#define 得了 }
#define 呀 ;
#include <iostream>
using namespace std;

那个啥 总的来说 我说 那个啥 啊 是吧
那么就 那个啥 有钱 等于 屁 呀
趁还 我说 有钱 是吧    那么就
那个啥 多少 呀 买 进 多少 呀 卖 出 多少 呀 有钱 去掉 多少 呀
卖 出 多少 呀 得了
拜拜了 啊 呀 得了
```

- Macro Names:
  1. #define PI_ROUNDED 3.14
  2. #define MIN(A,B) ((A) <= (B) ? (A) : (B)) //pay attention to the brackets
- Note
  - named with **all capitals and underscores**
- Example
  - int r1 = MIN(3,4);
  - float r2 = MIN(3.0, 4.0);

On Macro Names of Naming of ***Google C++ Style Guide***

Macro 宏

- Be very cautious with macros. 尽量别用
  - Prefer inline functions, enums, and const variables to macros.
  - Macros mean that the code you see is not the same as the code the compiler sees. (Not debugable)

- Luckily, macros are **not nearly as necessary** in C++ as they are in C.
  1. **Don't define macros in a .h file.**
  2. **#define macros right before you use them, and #undef them right after.**
  3. Do not just #undef an existing macro before replacing it with your own; instead, pick a name that's likely to be unique.

On Macro Names of Naming of ***Google C++ Style Guide***

Char & String

Char & string 字符和字符串

```
string strvar = "b";
char charvar = 'b';

cout << ('b' == charvar) << endl;
cout << ("b" == strvar) << endl;
//cout << ('a' == "a") << endl; // will error!
```

```
strvar = "b"
charvar = 'b'

print('b' == charvar)
print("b" == strvar)
print('a' == "a")
```

- C++: single quotes for char, & double quotes for string
- Python: single or double quotes.

String

- char helloworld[] = “Hello, world!”; // forget it
- string helloworld = “Hello, world!”; // use this
- char * chars = str.c_str(); // discuss it later

```
 1 #include <string>
 2 ...
 3 string s = "Hello";
 4 s += " world!";
 5 if(s == "Hello world!") {
 6      cout << "Success!" << endl;
 7 }
 8 cout << s.substr(6, 6) << endl; // Prints "world!"
 9 cout << s.find("world"); // (prints "6")
10 cout << s.find('l', 5); // (prints "9")
```

Variables变量

- There is also a special type which holds a memory location called **pointer**. Python not.

- C++ & Python also has **collection or compound data types**, which will be discussed later.

小结

- Types类型
  - int, float, boolean, char, string, …
- Consts常量
  - 无名常量：literals
  - 有名常量：const int i;
- Variables变量
  - **static typing vs. dynamic tying**
  - Declaration 显示声明 or not
- 起个好名字
- 诊断：**type() vs. typeid() & sizeof()**
- Operators 操作符
  - =, +, -, *, /, %
  - ==, <, >, && and, || or,
  - Compatible types & type conversions, Python not.

习题

1．若a为double性的变量，表达式a=1,a+5,a++的值为_ 1 _ 。

2．表达式7.5+1/2+45%10= 12.5 。

3．与! (x>2)等价的表达式是 x<=2 。

4．表达式于语句的重要区别是 (略) 。

5．x *= y+8等价于x= x*(y+8) 。