

User Id: weizhe.goh@digipen.edu Started: 2020.07.08 00:21:13 Score: 19%

DigiPen

Differences Between C And C++

Practice

© 2020, DigiPen Institute of Technology. All Rights Reserved

Introducing C++

- C++ is a general-purpose, object-oriented programming language designed by Bjarne Stroustrup in 1983.
- C++ shares a lot of C features. As a result, most of the C++ compilers can also compile C programs.
- Many other programming languages have been influenced by C++, including C#, Java, and newer versions of C.



Try to output words separately to produce the same result.

Outline

So why C++?

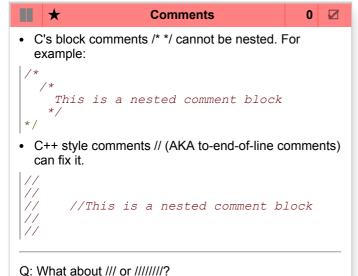
- · Makes programming easier
- · Makes the result safer
- · Overcomes C limitations
- Supports templated functions/structures/classes
- Has exception handling mechanism
- Supports object-oriented programming (OOP)

```
★ Hands-on
Compare the previous C++ style "Hello World!" example with old C version given below.
Run
```

```
#include <stdio.h>
int main() {
  print("%s","Hello C World!");
  return 0;
}

: In function 'int main()':
:4:3: error: 'print' was not declared in t
  4 | print("%s","Hello C World!");
  | ^~~~~
  | printf
```

Same as before, try to output words separately to produce the same result.



 Standard Input in C is received through scanf() function whereas standard output is given through printf() function.

```
scanf("%d", &n);
printf("Hello World!");
```

O Compile time error

O Still C++ style comment

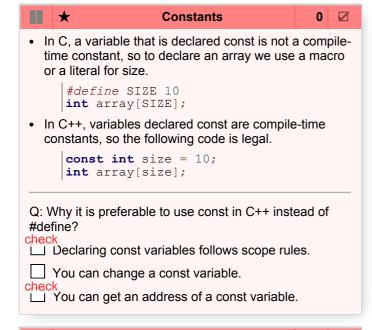
• C++ uses >> and << as standard input and output operators from std::cin and to std::cout respectively.

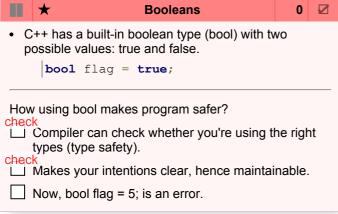
```
std::cin >> n;
std::cout << "Hello World!";</pre>
```

Hands-on

■ ★

```
Following code mixes C and C++ outputs. Make all
outputs C++ way.
 Run
#include <stdio.h>
#include <iostream>
int main()
   printf("%s ", "Hello");
   std::cout << "World!";</pre>
   return 0;
Hello World!
Edit the following C code to make the same output C++
way.
 Run
#include <stdio.h>
int main()
  printf("2+2=%d", 2+2);
  return 0;
2+2=4
```





```
Hands-on

Change the following code to use a constant variable instead of the macro definition.

Run

#include <stdio.h>
#define SIZE 10
int main()
{
  int array[SIZE] = { 5 };
  printf("%d", array[0]);
  return 0;
}
```

```
What is the result of a bool type variable output? Try out and choose your answer below.

Run
#include <iostream>int main()
{
    return 0;
}

One and zero

1 and 0

Not supported type

true and false
```

Prototypes

- In C, if the compiler reads a function call before its prototype or definition, it makes assumptions that return type is int and parameter types are the types of the arguments.
- Next function call assumes return type is int, 1st argument is int, 2nd argument is double:

```
fn(1, 3.0);
```

 In C++, this is illegal. Either function prototype or function definition must appear before the function can be called.

Declarations

- In C, all variables must be declared at the beginning of a scope.
- C++ allows declaring variables anywhere within the scope.

```
Hands-on 0 ☑
```

Change the following C code to C++ and defines all variables right before the use.

```
#include <iostream>
int main()
{
   int a = 2;
   std::cout << a << "+";
   int b = 3;
   std::cout << b << "=";
   int c = a + b;
   std::cout << c << "\n";
   return 0;
}

2+3=5
```

Can we or should we make declarations of a, b, and c as const?

Overloading

- It is not possible in C to overload an operator simbol (+ or %) of a function name.
- C++ allows such overloadings.
- The following C++ code shows the use of overloaded bitwise shift operators << as an example.

```
#include <iostream>
int main()
{
   std::cout << "Hello World!";
   return 0;
}</pre>
```

Make an example of a function name overloafing. Functions must have the same names, but must have different parameter types or numbers. Run Type your code here...

Conditional Operator

• In C, the conditional operator (?:) returns an r-value.

```
a = (b>c?b:c);
```

• In C++, it can be an I-value if both the second and third arguments are I-values of the same type.

```
(b>c?b:c) = 10;
```

```
★ Hands-on
Change the following code to an equivalent one that
```

Change the following code to an equivalent one that uses two conditional operators, one assignment and nothing else.

```
#include <iostream>
int main()
  int a=1, b=2, c=3, d=4;
  if (a > b)
     if (c > d)
       a = c;
     else
        a = d;
 else
     if (c > d)
        b = c;
     else
        b = d;
  std::cout << a << " " << b << " "
            << c << " " << d;
 return 0;
1 4 3 4
```

struct

Let's say we have a struct Foo.

• In C, declaration of a Foo type variable named w is:

```
struct Foo w;
```

• In C++, we can omit the struct tag.

```
Foo w;
```

· Same applies to union and enum

Hands-on

Create a struct A that has two members: first, p, is for the struct address, second, s, for the struct size. Initialize it and output in the main function.

```
#include <iostream>
struct A
{
  int p = 1;
  int s = 2;
};

int main()
{
  std::cout << A.p << "," << A.s;
  return 0;
}

: In function 'int main()':
:12:16: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected primary-expression
  12 | std::cout << A.p << "," << A.s;
  int main() /:
:12:30: error: expected prim
```

for loop

• C++ allows a declaration of variables with initialization in the for loop.

```
for (int i = 5, *pi = &i, ai[100] = {0}, j;
    i<10; ++) { }</pre>
```

• The scope of variables declared in this way ends at the end of the for loop body.

III ★ Hands-on

compilation terminated.

The following code is compilable by C++ compilers only. Change it to be compilable by C compilers as well.

```
#include <iostream>
#include <stdio.h>
int main()
{
   int i;
   for (i = 0; i < 3; i++)
       std::cout << i;
   for (i = 0; i < 3; i++)
       std::cout << i*i;
   return 0;
}

jdoodle.c:1:10: fatal error: iostream: No
   1 | #include</pre>
```

Headers

- C standard library header files (e.g., <time.h>) have their C++ counterparts (e.g., <ctime>).
 - The C++ version has no ".h" suffix but has an additional "c" prefix.
 - Note: <ctype.h> becomes <cctype>.
- Furthermore, in C++, all standard library functions should have std:: prefix, where std is a namespace, :: is a scope resolution operator.
 - E.g., the sqrt function from <cmath> has full name std::sqrt.

Hands-on 13

TBD

Typecasting

• In C, typecasting is done as follows:

```
int i = 5;
float f = (float) i / 2;
```

 This is still legal in C++, but C++ provides an alternative syntax for typecasting:

```
float f = float(i) / 2;
```

Hands-on

• Find equivalent expressions in the following code:

(Since C++11) long long

- long long is a new data type at least 64 bits in size.
 Capable of containing at least the

 [-9,223,372,036,854,775,808,
 +9,223,372,036,854,775,807] range
- AKA long long int, signed long long, signed long long int
- · Format specifier: %lli

(Since C++11) braced-init-list 1

braced-init-list is a new data type that can be used for:

· as the initializer in a variable definition

· as the initializer in a new expression

```
new std::vector<std::string>{"once",
    "upon", "a", "time"};
```

· in a return statement

```
return { "Norah" };
```

· as a function argument

```
f( {"Hello","World"} );
  // pass list of two elements
```

· and more ...

(Since C++11) braced-init-list 2

A braced-init-list may appear on the right-hand side of an

assignment to a scalar, in which case the initializer list must have at most a single element:

```
a = b = \{ 1 \}; // same as a = b = 1
```

• an assignment defined by a user-defined assignment operator, in which case the initializer list is passed as the argument to the operator function:

```
complex<double> z;
z = { 1,2 };
z += { 1, 2 };
```

(Since C++11) nullptr

- · nullptr is the pointer literal
- · Use nullptr to indicate that a pointer does not point to an object

```
int *pN = nullptr;
```

(Since C++11) auto

· auto is a new type specifier. Specifies that the type of the variable that is being declared will be automatically deduced from its initializer

```
auto a = 1 + 2;
std::cout << "type of a: "</pre>
      << typeid(a).name() << std::endl;
auto c = {1, 2};
std::cout << "type of c: "</pre>
             << typeid(c).name();
```

Output (GCC 7.2.0 and above):

```
type of a: i
type of c: St16initializer listIiE
```

Hands-on

(Since C++11) Range-based for loop

- · Executes a for loop over a range.
- · Used as a more readable equivalent to the traditional for loop operating over a range of values, such as all elements in an array.

```
int a[] = {0, 1, 2, 3, 4, 5};
for (int n : a)
  std::cout << n;</pre>
int a[] = \{0, 1, 2, 3, 4, 5\};
for (int &n : a)
  n++;
for (int n : a)
  std::cout << n;
```

| ★ Given the following code:

```
#include <iostream>
int main(void) {
   int a[] = { 1,
                   2, 3, 4, 5 };
    for (int i : a)
         i += 10;
    for (int i : a)
        std::cout << i << ' ';
   return 0;
```

What is the output?

- 11 12 13 14 15 O | Compilation error
- \bigcirc |1 2 3 4 5

Ø

0

(TBD) new/delete

- C++ has two operators new and delete that perform the task of allocating and freeing the memory in a better and easier way. More ... next time.
- TBD next class.

(TBD) References

- References, same as pointers, let you refer to other memory elements (like variables or structures/objects) indirectly.
- · TBD next class.

(TBD) Namespaces

- In C, every identifier declared at the global scope must have a unique name.
- In small programs, this is not a problem. But what about big projects, especially, that include libraries from other sources?
- C++ solution for this problem is to put identifiers in unique namespaces.
- You can create as many namespaces as you want, but one, named std, already existed in C++ and we use it for output in previous examples.

(TBD) Templates

- Templates are powerful features of C++ which allows you create, for example, a single function to work with different data types.
- Just one templated function can replace set of functions that are different only by data types. Below is a such example.

```
template<typename T>
void swap(T& x, T& y)
{
   T tmp = x;
   x = y;
   y = tmp;
}
```

 This template is equivalent to functions shown below and infinite number of similar functions with other data types.

```
void swap(int& x, int& y)
{
  int tmp = x;
  x = y;
  y = tmp;
}

void swap(bool& x, bool& y)
{
  bool tmp = x;
  x = y;
  y = tmp;
}
```

(TBD) OOP

- In C data and functions are separate elements of the code that does not provide an intuitive representation of reality.
- Object-oriented programming (OOP) provides us with the ability to create objects that tie together both properties (data) and behaviors (functions) into a selfcontained, reusable package called a *class*.
- The three pillars for OOP are discussed in details at the end of this course:
 - Encapsulation
 - Inheritance
 - Polymorphism

Read more

<u>Link</u> – New C++ features implemented in Microsoft C++ compiler are listed here

<u>Link</u> – You can compare new C++ features implemented in various compilers

 As you can see, Microsoft C++ compiler's support for new C++ is lagging behind other major compilers.

By signing this document you fully agree that all information provided therein is complete and true in all respects.

Responder sign:

Copyright © 2020 | Powered by MyTA | www.mytaonline.com