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

DigiPen

Differences Between C And C++

Practice

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.

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

Let's write our first "Hello World!" program C++ way.

Run

```
#include <iostream>

int main() {
  std::cout << "Hello C++ World!";
  return 0;
}
```

```
Hello C++ World!
```

Try to output words separately to produce the same result.

★ 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.

Comments 0

- C's block comments /* */ cannot be nested. For example:

```
/*
  /*
    This is a nested comment block
   */
*/
```

- C++ style comments // (AKA to-end-of-line comments) can fix it.

```
//
//
//    //This is a nested comment block
//
//
```

Q: What about /// or ////////?

- ◯ Compile time error
- ✔ ◯ Still C++ style comment

Hands-on 0

Following C code is not compilable. Let's fix it using //.

Run

```
#include <stdio.h>
int main()
{  /*
   while (i < count)
   {
       i++;
       /* printf("i is %d\n", i); */
   }
   */
   printf("Hello World!");
   return 0;
}
```

```
:9:5: error: expected unqualified-id befor
    9 |    */
      |     ^
:11:4: error: expected unqualified-id befo
   11 |    return 0;
      |    ^~~~~~
:12:1: error: expected declaration before
   12 | }
      | ^
```

I/O

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

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

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

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

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!";
   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
```

Constants 0

- In C, a variable that is declared const is not a compile-time constant, so to declare an array we use a macro or a literal for size.

```
#define SIZE 10
int array[SIZE];
```

- In C++, variables declared const are compile-time constants, so the following code is legal.

```
const int size = 10;
int array[size];
```

Q: Why it is preferable to use const in C++ instead of #define?

check
- [ ] Declaring const variables follows scope rules.
- [ ] You can change a const variable.

check
- [ ] You can get an address of a const variable.

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;
}
```

```
5
```

Booleans 0

- C++ has a built-in boolean type (bool) with two possible values: true and false.

```
bool flag = true;
```

How using bool makes program safer?

check
- [ ] Compiler can check whether you're using the right types (type safety).

check
- [ ] Makes your intentions clear, hence maintainable.
- [ ] Now, bool flag = 5; is an error.

Hands-on 0.5

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.

★ Hands-on ☐

Change the following code to make it compilable by C++ compiler.

Run

```
#include <iostream>
int main()
{
  print("Hello World!");
  return 0;
}
void print(const char* text)
{
  std::cout << text;
}
```

```
: In function ‘int main()’:
:4:3: error: ‘print’ was not declared in t
    4 |   print("Hello World!");
      |   ^~~~~
      |   printf
```

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.

Run

```
#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;
}
```

★ Hands-on

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 l-value if both the second and third arguments are l-values of the same type.

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

★ Hands-on

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

Run

```
#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.

Run

```
#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;
      |                ^
:12:30: error: expected primary-expression
   12 |  std::cout << A.p << "," << A.s;
      |                              ^
```

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; ++) { }
```

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

★ Hands-on

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

Run

```
#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
      |          ^~~~~~~~~~
compilation terminated.
```

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:

Run

```
#include <iostream>
int main() {
  std::cout << (float)3/2 << std::endl;
  std::cout << float(3)/2 << std::endl;
  std::cout << float(3/2) << std::endl;
}
```

```
jdoodle.c:1:10: fatal error: iostream: No
    1 | #include
      |          ^~~~~~~~~~
compilation terminated.
```

(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

```
T x3{a}; // direct initialization with
         //   braced-init-list
T x4 = { a };
T x5 = { }; // OK
```

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

- Output (GCC 7.2.0 and above):

```
type of a: i
type of c: St16initializer_listIiE
```

(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;
```

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

★ Hands-on 0 ☑

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`
- ○ `Compilation error`
- ✔ ○ `1 2 3 4 5`

(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 self-contained, 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

Link – New C++ features implemented in Microsoft C++ compiler are listed here

Link – 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: