0. 我的第一个 C++2z 程序：Helloworld

0. Helloworld

在之前的伪代码课程之中，我们借伪代码讲解了变量、函数与模块等概念。现在，让我们通过 Helloworld 来了解一下真实的 C++代码。

```
import std.io;
int main()
{
    std::cout << u8"Hello, world!" << std::endl;
}
```

将这段代码输入你喜欢的编译器中并运行，你将可以看到它的输出：

```
Hello, world!
```

Well done！你已经完成了你的第一个程序了。

1.从读代码开始

让我们来分析一下这段代码吧

```
import std.io;
int main()
{
    std::cout << u8"Hello, world!" << std::endl;
    return 0;
}
```

`import` *关键字*代表导入一个*模块*。`std.io` *模块*包含了主要的输入输出功能，下文中的 `std::cout` 与 `std::endl` 即来自于这个*模块*。

`int main(){…}` 是一个*函数定义*，它定义了一个返回类型为 `int`，名为 `main` ，无参数的*函数*。

*main 函数*是 C++中一个特殊的*函数*，一般代表程序的主入口点。

{} 函数主体用花括号包围。

`std::cout …;` 是一个语句，他完成了向屏幕输出并换行的功能。

`std::cout` 是来自于 `std.io` *模块*的一个*全局变量*，它以“流”的形式代表了程序的基本输出。 `std` 是一个*命名空间*，用以区分不同的名称。`cout` 是这个*变量*的变量名。

`<<` （ASCII+60）与 `+` 、 `-` 一样是一个*运算符*，代表了向“流”中插入数据。

`u8"Hello, world!"` 是一个*字符串字面量*，常常用来存储一段文本。
`u8` 表示字符串的编码为 UTF-8
`"…"` 由双引号（ASCII+34）括起来的文本就是*字符串字面量*的内容了。

`std::endl` 也是一个全局变量，用来表示刷新输出流并换行。

`;` 语句末尾以分号结束。

`return 0;` 返回语句。
`return` 是返回*关键字*。
`0` 为返回值，在 *main 函数*中返回 0 表示程序正常退出。

也许一些名词的细节你们现在还不了解，这些在接下来的课程中都将会有详细介绍。

1.C++的基础知识

0.基本数据类型

0.数据的内存表示

1.整数类型

`int` 是最基本的整数类型，至少有 16 位，在绝大多数的环境中都至少有 32 位。

对于整数类型，有两类修饰符：
符号性：
`signed` – 有符号（整数类型默认为有符号，如果省略）
`unsigned` – 无符号
宽度：
`short` – 至少为 16 位
`long` - 至少为 32 位
`long long` - 至少为 64 位
有修饰符的时候，`int` 可省略。修饰符与 `int` 的排列顺序任意。

例如，`unsigned long long int` 代表无符号至少为 64 位的整数类型，`signed short` 代表有符号至少 16 位的整数类型。`int long unsigned` 代表无符号至少 32 位的整数类型。

定宽整数

对于以上的类型，我们可以注意到对于他们对宽度的描述，都有“至少”这样的描述，这表示他们的宽度是由具体实现决定的。在某些需要一个确定宽度的场合，我们则需要一系列定宽类型。

定宽类型具有以下形式

`[u]int{8, 16, 32, 64}_t` ，特定位数的、无/有符号的整数类型。u 表示是否有符号，数值代表特定位数。如 `uint64_t` 表示 64 位无符号数，int16_t 表示 16 位有符号数。特别地，对于有符号数，定宽整数没有填充位且由补码实现。某些平台可能不完全支持这些类型。

除此之外，还有一些针对特定宽度的整数类型

`[u]int_least{8, 16, 32, 64}_t` 最小的至少拥有特定位数的无/有符号整数类型。

`[u]int_fast{8, 16, 32, 64}_t` 最快的至少拥有特定位数的无/有符号整数类型。

`[u]intmax_t` 平台支持的最大的无/有符号整数类型

特殊整数类型

常见的整数类型还有 `size_t` 与 `intptr_t` 、`ptrdiff_t` 。

`size_t` 常用来表示大小、数组下标等等的非负整数。

`[u]intptr_t` 可以容纳一个指针的无/有符号整数类型。

`ptrdiff_t` 用来表示两个指针之差的有符号整数类型。

整数类型的范围

整形字面量

整数类型的选择

2.字符类型

0) `char` 字符型，用来表示系统中的字符，是 C++里宽度最小的类型。`char` 至少可以表示 256 个不同的值（常见的如[0, 255]或者[-128, 127]）。
1) `signed char` 代表有符号字符
2) `unsigned char` 代表无符号字符。

`char` 必然与 `signed char` 或 `unsigned char` 之一有相同的表示方法（但它们是三个独立的类型）。在多数环境中，它们的宽度都是 8 bit = 1 byte ，即八位二进制，其中有符号的数以补码表示。大多数环境的内存最小单元也是 8 bit = 1 byte，这使得 `char` 特别是 `unsigned char` 可以作为直接表示内存、对象结构的类型。

3) `char16_t` UTF-16 字符型，至少有 16 位的无符号数。
4) `char32_t` UTF-32 字符型，至少有 32 位的无符号数。

对于这些整数型和字符型的宽度，我们有

```
1 == sizeof(char) <= sizeof(short) <= sizeof(int) <= sizeof(long) <= sizeof(long long)
```

3.布尔类型

`bool` 是一种用来存储布尔值的类型。它只有两个值：`true` 与 `false` 。在转换为整数类型时，`true` 会转换为 1，`false` 则会转换为 0。

4.浮点类型

浮点数即我们常说的小数，有以下三种类型。
`float` 单精度浮点类型。通常是符合 IEEE-754 标准的 32 位浮点数。

`double` 双精度浮点类型。通常是符合 IEEE-754 标准的 64 位浮点数。

`long double` 扩展精度浮点类型。多数为 80 位浮点数。

浮点类型的范围

浮点字面量

浮点类型的选择

5.类型的自动推导

1.运算符

2.表达式

常量表达式

3.语句

0.表达式语句

将表达式末尾加上分号；（ASCII+59），就成为了一个表达式语句。特别地，单独的一个分号 ；可以看作一个空语句。例如以下每一行都是一个表达式语句：

```
a + b;
f(a);
x = a - b;
std::cin >> a;
;
```

1.复合语句

将多个语句由{} （ASCII+123 125）包围起来，就形成了复合语句。例如

```
{
    auto x = 0;
    int a, b;
    std::cin >> a >> b;
    x = a - b;
    std::cout << x;
}
```

这个整体就是一个复合语句。

2.选择执行语句

`if` 与 `switch` 可以被归类为选择执行语句。它们可以根据条件选择是否执行、执行哪个语句的能力。

`if` 语句

`if` 语句用来根据条件选择一个语句是否执行（单独使用 `if`），或者从两个语句中选择一个执行（使用 `if … else …`）。

以下是一些 `if` 语句的例子：

```
if constexpr (constexpr int i = 5 + 6;i > 10) {
    std::cout << u8"5 + 6 = "  << i << u8", which is greater than 10! " << std::endl;
}
```

```
if (a > b) a = b;
else {
    auto c = a;
    a = -b;
    b = c;
}
```

```
if (int diff = a - b;a > b) c = a;
else {
    if (a < b) c = diff;
    else c = b;
}
```

```
if (int diff = a - b) c = b;
else {
    if (diff < 0) c = diff;
    else c = a;
}
```

`if` 语句拥有以下形式：

`if constexpr`可选 (声明语句可选 条件) 语句

以及

`if constexpr`可选 (初始化语句可选 条件) 语句 `else` 语句

如果条件为真（即为 `true` 或可以转换为 `true`），则 `if` 后的语句被执行，如果为假则不执行或执行 `else` 后的语句。

例如

```
auto a = 0, b = 1;
if (a > b) a = b;
```

中 `a = b;` 将不会被执行。

括号中还可以包含一个初始化语句，可用来定义变量或者执行任意表达式。

例如

```
int32_t a = 2, b = 1, c;
if ( auto diff =  a - b; diff > 0) c = diff;  else c = - diff;
```

其中，初始化语句定义了变量 diff 并初始化为 a – b 即 1 ，条件 diff > 0 成立，执行 c = diff;

注意条件本身也可为一个声明（如 `if (int diff = a - b)`），这使得我们可以写出这样的 `if` 语句：

```
if (int32_t diff = a - b; uint32_t diff_abs = diff > 0 ? diff : -diff){
    std::cout << u8"a != b and"
              << u8" a - b is " << diff
              << u8" abs(a - b) is " << diff_abs
              << std::endl;
}
```

对于 `if … if … else …` 这种情形，我们规定这个 `else` 属于最近的第一个还没有 `else` 的 `if` 语句。

如果 `if` 语句是 `if constexpr` 的形式，则要求条件是一个常量表达式。

`switch` 语句

`switch` 语句用来根据条件，转移到某一个语句执行。

以下是一些 `switch` 语句的例子：

```
switch (a - b) {
case 0:
    std::cout << "a - b = 0" << std::endl;
    break;
case 1:
    std::cout << "a - b = 1" << std::endl;
    break;
case 2:
    std::cout << "a - b = 2" << std::endl;
    break;
case 3:
    std::cout << "a - b = 3" << std::endl;
    [[fallthrough]];
case 4:
    std::cout << "a - b = 3 or 4" << std::endl;
    break;
default :
    std::cout << "a - b fall out of range of [0, 4]" << std::endl;
}
```

```
switch (int x = a) {
case 0:
    [[fallthrough]];
case 1:
    [[fallthrough]];
case 2:
    [[fallthrough]];
case 3:
    std::cout << x << "belongs to [0, 3]" << std::endl;
    break;
case 4:
    std::cout << x << " = 4" << std::endl;
    break;
default:
    std::cout << "x fall out of range of [0, 4]" << std::endl;
}
```

`switch` 语句拥有以下形式：

```
switch （声明语句可选 条件）语句
```

其中，语句中可包含 `case` 标签与 `default` 标签和 `break` 语句，其中 `case` 标签与 `default` 标签有以下形式：

```
case 常量表达式: 语句
```

```
default : 语句
```

`switch` 语句根据条件，跳转到对应的 `case` 标签（如果有）或 `default` 标签开始执行，直到结束或遇到 `break` 语句。`switch` 语句允许跨过 `case` 标签执行（如上例 0-3 都会执行至 `std::cout << x << "belongs to [0, 3]" << std::endl;`），即“fallthrough“。但大多数编译器都会在没有显式提供`[[fallthrough]]` 时发出警告。

声明语句和条件的要求与 `if` 语句基本一致。

3.循环语句

`while` 语句

`while` 语句用于条件不为 `false` 时循环执行语句。

以下是一些 `while` 语句的例子：

```
int a = 20;
while (a > 0) std::cout << a-- << std::endl;
```

```
int a = 20;
while (int b = f(a)) {
```

```
    std::cout << b << std::endl;
}
```

`while` 语句拥有以下形式：

```
while (条件) 语句
```

如果条件不为 `false`，则执行语句并再次开始循环。即每次执行语句前检查条件，条件为真就最新语句并再次检查，条件为假则跳出。

与 `if` 语句等相似，`while` 语句的条件也可以是一个声明，但应注意的是这个声明是对应于特定的迭代的，即每次执行条件都是一个独立的声明。

`break` 语句可用于跳出 `while` 语句，结束整个循环。

```
int a = 20;
while (a > 0) {
    std::cout << a << std::endl;
    if(a-- == 10) break;
}
```

`continue` 语句可用于提前结束一次循环。

```
int a = 20;
while (a > 0) {
    if(a % 2) continue;
    std::cout << a-- << std::endl;
}
```

do 语句

`do` 语句(do…while)用来循环执行语句直到条件为 `false`。

以下是 `do` 语句的例子：

```
int a = 0;
do{
    std::cout << a-- << std::endl;
} while (a > 0)
```

```
int a = 20;
do{
    if(a % 2) continue;
    std::cout << a-- << std::endl;
} while (a > 0)
```

`do` 语句拥有以下形式：

```
do 语句 while (表达式);
```

do 语句与 while 语句的主要不同在于它在循环结束后判断表达式的值，如果为真则继续循环，即必定执行一次循环。

break 语句与 continue 语句的作用与 while 语句中相似。

for 语句（基本形式）

执行初始化语句，条件不为 false 时循环执行循环体与迭代表达式。

以下是 for 语句的例子：

```
for(int a = 20; a > 0; --a){
    if(a % 2) continue;
    std::cout << a << std::endl;
}
```

```
for(int a = 20;; --a){
    if(a % 2) continue;
    std::cout << a << std::endl;
    if(a == 1) break;
}
```

for 语句拥有以下形式：

```
for (声明语句 条件可选; 迭代表达式可选) 语句
```

其中，声明语句中的声明可以覆盖所有循环，即在 for 语句中只在最开头执行一次。

条件与迭代表达式都是可选的，且条件为空时默认为真

如果条件与迭代表达式都不选，而声明语句为空语句时，for 语句如

```
int a = 20;
for(;;){
    std::cout << a << std::endl;
    if(a == 1) break;
    --a;
}
```

4.跳转语句

break 语句与 continue 语句

这两个语句作用于循环语句与 `switch` 语句，在对应的语句中分别有介绍。

goto 语句

`goto` 语句用于跳转到特定的标签执行。

```
int a = 10;
label:
    std::cout << a <<" ";
    a = a - 2;

    if (a != 0) {
        goto label;
    }
    std::cout << '\n';

    for (int x = 0; x < 3; x++) {
        for (int y = 0; y < 3; y++) {
            std::cout << "(" << x << ";" << y << ") " << '\n';
            if (x + y >= 3) {
                goto endloop;
            }
        }
    }
endloop:
    std::cout << std::endl;
```