assert.h

目录

assert()

static_assert()

assert()
assert.h 头文件定义了宏 assert()
，用于在运行时确保程序符合指定条件，如果不符合，就报错终止运行。这个宏常常被称为 “断言 ”。

assert(PI > 3);

上面代码在程序运行到这一行语句时，验证变量 PI是否大于 3。如果确实大于 3
，程序继续运行，否则就会终止运行，并且给出报错信息提示。

assert() 宏接受一个表达式作为参数。如果该表达式为真（返回值非零）， assert()
不会产生任何作用，程序继续运行。如果该表达式为假（返回值为零）， assert()就会报错，在标准错误流 stderr中写入一条错误信息，显示没有通过的表达式，以及包含这个表达式的文件名和行号。最后，调用
abort() 函数终止程序（ abort()函数的原型在 stdlib.h头文件中）。

z = x * x - y * y;
assert(z >= 0);

上面的 assert()语句类似于下面的代码。

if (z < 0) {
puts("z less than 0");
abort();
}

如果断言失败，程序会中断执行，会显示下面的提示。

Assertion failed: (z >= 0), function main, file /Users/assert.c, line 14.

上面报错的格式如下。

```
Assertion failed: [expression], function [abc], file [xyz], line [nnn].
```

上面代码中，方括号的部分使用实际数据替换掉。

使用 `assert()` 有几个好处：它不仅能自动标识文件和出问题的行号，还有一种无需更改代码就能开启或关闭 `assert()` 的机制。如果已经确认程序没有问题，不需要再做断言，就在 `#include <assert.h>` 语句的前面，定义一个宏 `NDEBUG` 。

```
#define NDEBUG
#include <assert.h>
```

然后，重新编译程序，编译器就会禁用文件中所有的 `assert()` 语句。如果程序又出现问题，可以移除这条 `#define NDBUG` 指令（或者把它注释掉），再次编译，这样就重新启用了 `assert()` 语句。

`assert()` 的缺点是，因为引入了额外的检查，增加了程序的运行时间。

static_assert()

C11 引入了静态断言 `static_assert()` ，用于在编译阶段进行断言判断。

```
static_assert(constant-expression, string-literal);
```

`static_assert()` 接受两个参数，第一个参数 `constant-expression` 是一个常量表达式，第二个参数 `string-literal` 是一个提示字符串。如果第一个参数的值为 false ，会产生一条编译错误，第二个参数就是错误提示信息。

```
static_assert(sizeof(int) == 4, "64-bit code generation is not supported.");
```

上面代码的意思是，如果当前计算机的 `int` 类型不等于 4个字节，就会编译报错。

注意， `static_assert()` 只在编译阶段运行，无法获得变量的值。如果对变量进行静态断言，就会导致编译错误。

```
int positive(const int n) {
  static_assert(n > 0, "value must > 0");
  return 0;
}
```

上面代码会导致编译报错，因为编译时无法知道变量 `n` 的值。

static_assert() 的好处是，尽量在编译阶段发现错误，避免运行时再报错，节省开发时间。另外，有些
assert() 断言位于函数之中，如果不执行该函数，就不会报错，而 static_assert() 不管函数是否执行，都会进行断言判断。最后， static_assert()
不会生成可执行代码，所以不会造成任何运行时的性能损失。

版本号 #2
由 壳壳虫 创建于 2021-09-04 22:34:34 UTC
由 壳壳虫 更新于 2021-11-02 10:47:55 UTC