

2020 年全國 C++ 等級考試

試題解析冊

(一)、選擇題

1. 略

2. 解析：

- A 選項考慮位元運算與取記憶體位址
- B 選項考慮比較運算與 `template <>`
- C 選項考慮 `new` 運算子與 `placement new`
- D 選項考慮位元運算與解構子
- E 選項沒有多個含義
- F 選項考慮成員訪問與小數點
- G 選項沒有多個含義

3. 解析：

- A 選項支援巢狀，甚至支援多層巢狀
- B 選項不支援巢狀
- C 選項不支援巢狀
- D 選項不支援巢狀，《C++ Primer》中明確提及
- E 選項支援巢狀
- F 選項支援巢狀，考慮 `extern {extern {...}}`，《C++ Primer》中有明確提及
- G 選項與 A 選項類似，支援巢狀

4. 解析：

被 `constexpr` 標識的變數自帶 `const` 屬性，因此 `i` 的型別為 `const int`

A 選項由 `decltype(i)` 推導而來的型別為 `const int &`，最外層的括弧起干擾作用，一個參考必須在宣告時就被初始化

B 選項由 `decltype(i)` 推導而來的型別為 `const int`，於是 `i3` 的型別為 `const int &&`，只能使用右值對 `i3` 進行初始化，因此選項正確

C 選項為 C++ 內建的初始化種類之一

D 選項 `&i` 的型別為 `const int *`，它只能隱含地向 `const void *` 轉型，若要轉型至 `void *`，需要使用 `const_cast` 去除頂層 `const`

E 選項 `i5` 的型別為 `int &`，若要參考至 `i`，那麼 `i5` 的型別應該為 `const int &`，需要使用 `const_cast` 去除頂層 `const`。本選項中的 `static` 起干擾作用

F 選項 `i` 是 `const int` 型別，它可以隱含地向另外一個內建的整型轉化，因此本選項正確

G 選項不具名列舉中的常數會污染外層的可視範圍，而外層已經存在一個 `i`，因此會產生重名而導致編碼錯誤

5. 解析：

- A 選項 C++ 11 引入
- B 選項由 C 繼承到 C++
- C 選項 C++ 98 引入
- D 選項 C++ 98 引入
- E 選項由 C 繼承到 C++，C++ 11 賦予全新意義
- F 選項 C++ 98 引入
- G 選項 C++ 11 中非關鍵字，C++ 20 引入

6. 解析：

A 選項 `nullptr` 的型別實際為 `nullptr_t`，因此不可以使用 `reinterpret_cast` 進行轉型，應該使用 `static_cast`

B 選項任意成員函式都可以是被刪除的函式

- C 選項 auto 不支援列表初始化
- D 選項 throw 可以出現在被 noexcept 標識的函式中，某些編碼器會因此發出警告
- E 選項 auto 不支援陣列的推斷，需要明確指明 `int arr[] = {1, 2, 3};`
- F 選項考慮 `T = void`
- G 選項 a 為變數，在前處理階段編碼器不會講變數與巨集名稱混合

7. 解析：

- A 選項圓括弧不支援列表初始化
- B 選項 `std::vector` 的擴充會導致疊代器失效，從而對已經被回收的記憶體讀取
- C 選項對於一個物件來說，`sizeof` 運算子可以省略圓括弧
- D 選項 `delete this;` 發生時，必須保證類別不會被二次回收
- E 選項考慮指標的比較
- F 選項當 d 是由 `new` 運算子產生的物件時，Base 類別的解構子沒有設定為虛擬函式時，才會發生未定行為。僅有一個 `;` 的陳述式是合法的，因此 `Derived d;;` 沒有任何問題
- G 選項當 `placement new` 對某個記憶體位址建構之後，使用非 `placement new` 回傳的指標會產生未定行為

8. 解析：

- A 選項 `std::vector` 與 `std::string` 的陣列注標運算子只支援整型，當給定的整型引數超出 `std::vector` 或 `std::string` 的範圍會發生未定行為
- B 選項 `std::map` 與 `std::set` 使用其陣列注標運算子查詢不存在的元素時，會在容器內創建一個這樣的元素
- C 選項使用尾置回傳型別可以回傳一個陣列的參考
- D 選項對陣列型別的推導，`decltype` 及 `auto` 會有不同的結果。`decltype` 推導的結果是 `T [N]`，`auto` 推導的結果是 `T *`
- E 選項正確
- F 選項 `priority_queue` 不可以被列表初始化
- G 選項不具名 `union` 可以像不具名 `namespace` 或不具名 `enum` 一樣直接存取其成員

(二)、填空題

9. 解析：

由 `i3` 不可被初始化可以得到 `i3` 是非參考型別，由於 `i2` 是參考型別，要得到一個非參考型別，只要得到一個右值，再使用 `decltype` 對這個右值進行推導即可

10. 解析：

`constexpr` 及 `inline` 並不影響型別。另外，`void ()` 是函式型別，`void` 並非函式型別。除此之外，由於不存在 `void &` 型別，因此 `typename add_lvalue_reference<void>::type` 的結果仍然為 `void`，因此最終 `noexcept` 表達式的結果為 `noexcept(true)`，內部的 `noexcept` 是運算子，外部的 `noexcept` 是標識符

11. 略

12. 解析：

對於參數為任意大小的陣列，考慮使用 `template <size_t N>` 推導陣列大小，再將參數設定為陣列的參考 `T (&)[N]` 即可

13. 解析：

對於名稱空間中的名稱搜尋，有一個重要的例外：如果一個函式接受的引數為一個類別型別，首先會在當前可視範圍內搜尋，之後在外層可視範圍內搜尋，接著還會去查找引數類

別對應型別的可視範圍的名稱。這一例外對於參考與指標同樣適用。這條例外帶來的好處，就是我們可以對某個型別直接使用在其名稱空間之下的運算子

14. 解析：

NULL 不同於 nullptr，它是巨集，用於替換數字 0。為了解決它在函式呼叫中模稜兩可的問題，C++ 引入了 nullptr。而 0 可以是一個數字，也可以是一個空指標，對於前三個宣告來說，0 都可以向其進行一次隱含型別轉換。只有前三個多載函式的候選都不可行，編碼器才會選擇第四個多載函式

(三)、改錯題

15. 解析：

- (1)、類別內對成員變數進行預設初始化的時候，不可以使用圓括弧，只能使用花括弧
- (2)、要在類別內直接初始化靜態成員變數，要求對其標識 constexpr
- (3)、合成宣告 (= default) 只能用於預設建構子、複製建構子、移動建構子、複製指派運算子、移動指派運算子及解構子，用於其它函式會產生編碼錯誤
- (4)、若某個類別擁有虛擬建構子，那麼編碼器不會為其合成移動建構子
- (5)、衍生類別必須顯式地初始化其基礎類別
- (6)、override 在 C++ 中既可以是一個關鍵字也可以被作為變數和函式等的名稱
- (7)、Bar b2 = static_cast<Bar &&>(b1); 當移動建構有效時，使用的是移動建構子，否則將使用複製建構子

16. 解析：

- (1)、不存在 void & 型別
- (2)、不存在函式的參考型別
- (3)、C++ 中為陣列引入了陣列的參考

17. 解析：

- (1)、decltype 對函式的推導得到的只是函式型別並非函式指標型別
- (2)、可以使用 const_cast 去除底層 const，但是對去除之後的指標或著參考進行寫入是未定行為
- (3)、局域類別不可以獲取到函式中的局域變數
- (4)、多層可視範圍內存在同名變數覆蓋，優先取用離使用處最近的可視範圍內同名變數

18. 解析：

- (1)、遞迴函式必須有結束標誌
- (2)、一個表達式中多次出現同一個變數，C++ 不規定對這個變數的操作順序。也就是例如 f(++n, n)，編碼器可以先放入 n 再執行 ++n，最後放入 ++n 的結果；同時，編碼器可以首先執行 ++n，然後放入結果
- (3)、lambda 要使用外部的變數必須捕獲
- (4)、lambda 預設是一個可呼叫物件，如果不呼叫，就不產生結果
- (5)、對 bool 變數的取反操作必須使用 ! 運算子，使用 - 運算子對 bool 型別進行運算時，bool 型別首先轉型為 int 型別。當 symbol 為 true 時，symbol 轉型為 1，使用 - 運算子使得 1 轉換為 -1，但是 -1 轉型為 bool 型別時仍然是 true；當 symbol 為 false 時，同理，永遠為 false
- (6)、C++ 11 中規定多行的 lambda 表達式預設回傳型別為 void
- (7)、當 n 足夠大，階乘產生的結果會超出 int 所容納的範圍，因此必須重新考慮演算法，考察前項與後項之間的關係

(四)、論述題

解析略，見標準答案

(五)、程式設計題

解析略，見標準答案與評分標準