Casting operators in C++ | Set 1 (const_cast)

C++ supports following 4 types of casting operators:

1. const_cast
2. static_cast
3. dynamic_cast
4. reinterpret_cast

1. const_cast
const_cast is used to cast away the constness of variables. Following are some interesting facts about const_cast.

1) const_cast can be used to change non-const class members inside a const member function. Consider the following code snippet. Inside const member function fun(), 'this' is treated by the compiler as 'const student* const this', i.e. 'this' is a constant pointer to a constant object, thus compiler doesn't allow to change the data members through 'this' pointer. const_cast changes the type of 'this' pointer to 'student* const this'.

```
#include <iostream>
using namespace std;

class student
{
private:
    int roll;
public:
    // constructor
    student(int r):roll(r) {}

    // A const function that changes roll with the help of const_cast
    void fun() const
    {
        ( const_cast <student*> (this) )->roll = 5;
    }

    int getRoll()  { return roll; }
};

int main(void)
{
    student s(3);
    cout << "Old roll number: " << s.getRoll() << endl;

    s.fun();

    cout << "New roll number: " << s.getRoll() << endl;

    return 0;
}
```

Output:

```
Old roll number: 3
New roll number: 5
```

2) const_cast can be used to pass const data to a function that doesn't receive const. For example, in the following program fun() receives a normal pointer, but a pointer to a const can be passed with the help of const_cast.

```
#include <iostream>
using namespace std;

int fun(int* ptr)
```

```
{
    return (*ptr + 10);
}

int main(void)
{
    const int val = 10;
    const int *ptr = &val;
    int *ptr1 = const_cast <int *>(ptr);
    cout << fun(ptr1);
    return 0;
}
```

Output:

```
20
```

3) It is undefined behavior to modify a value which is initially declared as const. Consider the following program. The output of the program is undefined. The variable 'val' is a const variable and the call 'fun(ptr1)' tries to modify 'val' using const_cast.

```
#include <iostream>
using namespace std;

int fun(int* ptr)
{
    *ptr = *ptr + 10;
    return (*ptr);
}

int main(void)
{
    const int val = 10;
    const int *ptr = &val;
    int *ptr1 = const_cast <int *>(ptr);
    fun(ptr1);
    cout << val;
    return 0;
}
```

Output:

```
 Undefined Behavior
```

It it fine to modify a value which is not initially declared as const. For example, in the above program, if we remove const from declaration of val, the program will produce 20 as output.

```
#include <iostream>
using namespace std;

int fun(int* ptr)
{
```

```
    *ptr = *ptr + 10;
    return (*ptr);
}

int main(void)
{
    int val = 10;
    const int *ptr = &val;
    int *ptr1 = const_cast <int *>(ptr);
    fun(ptr1);
    cout << val;
    return 0;
}
```

4) const_cast is considered safer than simple type casting. It'safer in the sense that the casting won't happen if the type of cast is not same as original object. For example, the following program fails in compilation because 'int *' is being typecasted to 'char *'

```
#include <iostream>
using namespace std;

int main(void)
{
    int a1 = 40;
    const int* b1 = &a1;
    char* c1 = const_cast <char *> (b1); // compiler error
    *c1 = 'A';
    return 0;
}
```

output:

```
prog.cpp: In function 'int main()':
prog.cpp:8: error: invalid const_cast from type 'const int*' to type 'char*'
```

5) const_cast can also be used to cast away volatile attribute. For example, in the following program, the typeid of b1 is PVKi (pointer to a volatile and constant integer) and typeid of c1 is Pi (Pointer to integer)

```
#include <iostream>
#include <typeinfo>
using namespace std;

int main(void)
{
    int a1 = 40;
    const volatile int* b1 = &a1;
    cout << "typeid of b1 " << typeid(b1).name() << '\n';
    int* c1 = const_cast <int *> (b1);
    cout << "typeid of c1 " << typeid(c1).name() << '\n';
```

```
    return 0;
}
```

Output:

```
typeid of b1 PVKi
typeid of c1 Pi
```

Exercise
Predict the output of following programs. If there are compilation errors, then fix them.

Question 1

```
#include <iostream>
using namespace std;

int main(void)
{
    int a1 = 40;
    const int* b1 = &a1;
    char* c1 = (char *)(b1);
    *c1 = 'A';
    return 0;
}
```

Question 2

```
#include <iostream>
using namespace std;

class student
{
private:
    const int roll;
public:
    // constructor
    student(int r):roll(r) {}

    // A const function that changes roll with the help of const_cast
    void fun() const
    {
        ( const_cast <student*> (this) )->roll = 5;
    }

    int getRoll()  { return roll; }
};

int main(void)
{
    student s(3);
    cout << "Old roll number: " << s.getRoll() << endl;
```

```
    s.fun();

    cout << "New roll number: " << s.getRoll() << endl;

    return 0;
}
```

–Aashish Barnwal. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Related Topics:

- Comparison of a float with a value in C
- Pure virtual destructor in C++
- C++ mutable keyword
- Is it possible to call constructor and destructor explicitly?
- Multithreading in C
- How to print range of basic data types without any library function and constant in C?
- C++ final specifier
- Print substring of a given string without using any string function and loop in C

Writing code in comment? Please use **ideone.com** and share the link here.

c++ it gives the output:20 without undefined behaviour ...
please check it out. and let me know if I am wrong.

Reply • Share ›

Subbu • 8 months ago

I guess for example 2
int main(void)
{
const int val = 10;
const int *ptr = &val;
int *ptr1 = const_cast <int *="">(ptr);
will cause compilation errors .it should be
int val = 10;
const int *ptr = &val;
int *ptr1 = const_cast <int *="">(ptr);

Reply • Share ›

gee • a year ago

this discussion only contains "const_cast". please detail out about other casting methods mentioned.

1 Reply • Share ›

Palaniselvam Sivasamy • 2 years ago

const_cast<int>(&(const_cast <student*> (this))->roll) = 5;.

The above code can solve for Question 2.

Reply • Share ›

Bibek ➚ Palaniselvam Sivasamy • 4 months ago

can u plz explain ?

Reply • Share ›

shan ➚ Bibek • a month ago

1) first inner const_cast is used to remove constness of this pointer
2) after accessing this->roll compiler will give error because roll is const so we have to remove this const
3) & is used because const_cast will only work on pointer
4) & gives address of this->roll and now we can cast away costness and assign values.

Reply • Share ›

Palani • 2 years ago

*const_cast(&(const_cast (this))->roll) = 5;

The above code can solve for Question 2

Reply • Share ›

Mayank Gupta • 2 years ago

void fun() const
{
(const_cast (this))->roll = 5;
}

In this snippet, I think the typecasting of (const student* const this) is to (student* this) and not (student* const this).

Reply • Share ›

Aashish ➚ Mayank Gupta • 2 years ago

this pointer would be still constant. If it would have been the case, you would have been able to change this pointer to point to some other object. ARe you able to do so?
So, it would be (student* const this).

Reply • Share ›

Sandeep • 2 years ago

From Question 2 we can conclude that, const_cast cannot be used to change non-const class members.

Reply • Share ›

Sandeep ➚ Sandeep • 2 years ago

const class members*

1 Reply • Share ›

Sandeep ➚ Sandeep • 2 years ago

const class members

Reply • Share ›

SSR • 2 years ago

program 3 is giving 10 as output on my machine . gcc complier.but not undefined.

Reply • Share ›

ravi ➚ SSR • 2 years ago

Dude, undefined behavior means the output is compiler dependent. It doesn't mean that you will get "undefined" on screen :)

Reply • Share ›

ssr ➚ ravi • 2 years ago

thanks for clearing that up kind of new in this thing :)

Reply • Share ›

- Interview Experiences
- Advanced Data Structures
- Dynamic Programming
- Greedy Algorithms
- Backtracking
- Pattern Searching
- Divide & Conquer
- Mathematical Algorithms
- Recursion
- Geometric Algorithms

-

- # Popular Posts

  - All permutations of a given string
  - Memory Layout of C Programs
  - Understanding “extern” keyword in C
  - Median of two sorted arrays
  - Tree traversal without recursion and without stack!
  - Structure Member Alignment, Padding and Data Packing
  - Intersection point of two Linked Lists
  - Lowest Common Ancestor in a BST.
  - Check if a binary tree is BST or not
  - Sorted Linked List to Balanced BST

- Follow @GeeksforGeeks Subscribe

- # Recent Comments

  - RK

    Dude please do not make comments section dirty....

    Amazon Interview Experience | Set 176 (For SDE 1) · 41 minutes ago

  - Rasmi Ranjan Nayak

    I did not understand the round-1 Question...

    Amazon Interview Experience | Set 175 (For SDE) · 1 hour ago

  - neer1304

    void merge(int A[], int m, int B[], int n) {...

    Merge an array of size n into another array of size m+n · 2 hours ago

  - darkprotocol

    I would suggest to use while loop instead or...

    Function to check if a singly linked list is palindrome · 3 hours ago

  - amit jaiswal

    static int k=3; void kthsmallest(struct tree *...

    Find k-th smallest element in BST (Order Statistics in BST) · 3 hours ago

  - tejavadali

    bool fixNumber(int a[], int ci, int n) { int...

    Rearrange array in alternating positive & negative items with O(1) extra space · 4 hours ago