Below is a **complete, copy-and-paste** MFC implementation that turns any ordinary `CEdit` into a “numeric-only” edit box.
The class:

* derives from `CEdit`;
* overrides `PreSubclassWindow()` so it can hook the control’s window procedure;
* handles the `WM_CHAR` message (or you could use `OnChar`) to filter input;
* optionally allows **negative numbers** and a single decimal point.

You can drop this header / source pair into any MFC project, declare your edit controls with the *class name* of `CNumberEdit`, and you’re done.

---

1. Header – `NumberEdit.h`

```cpp
#pragma once
#include <afxwin.h>
class CNumberEdit : public Cedit
{
public:
    // Allow both integers and decimals, optionally negative.
    enum class EFlags
    {
        None          = 0x00,
        AllowNegative = 0x01,   // e.g. "-123"
        AllowDecimal  = 0x02    // e.g. "12.34"
    };

    CNumberEdit(EFlags flags = EFlags::None);

protected:
    afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
    DECLARE_MESSAGE_MAP()

private:
    bool IsValidChar(char ch) const;
    void HandleBackspace();
    void InsertChar(char ch);

    EFlags m_flags;
};
```

---

2. Implementation – `NumberEdit.cpp`

```cpp
#include "NumberEdit.h"

CNumberEdit::CNumberEdit(EFlags flags)
    : m_flags(flags)
```

```
{
}

BEGIN_MESSAGE_MAP(CNumberEdit, CEdit)
    ON_WM_CHAR()
END_MESSAGE_MAP()

void CNumberEdit::OnChar(UINT nChar, UINT /*nRepCnt*/, UINT /*nFlags*/)
{
    // Allow control keys (e.g. TAB, ENTER, ESC) to pass through unchanged.
    if (nChar < 32 || nChar == VK_RETURN || nChar == VK_ESCAPE)
    {
        CEdit::OnChar(nChar, 0, 0);
        return;
    }

    // Backspace or Delete – we just let the base class handle it
    if (nChar == VK_BACK || nChar == VK_DELETE)
    {
        HandleBackspace();
        return;
    }

    char ch = static_cast<char>(nChar);

    if (!IsValidChar(ch))
    {
        // beep to indicate invalid input
        MessageBeep(MB_ICONHAND);
        return;          // swallow the character
    }

    InsertChar(ch);      // we manually insert because we want to control caret
placement
}

// ---------------------------------------------------------------------------
// Helper: decide whether a char is allowed given the flags and current text.
// ---------------------------------------------------------------------------
bool CNumberEdit::IsValidChar(char ch) const
{
    CString txt;
    GetWindowText(txt);

    // digits are always allowed
    if (ch >= '0' && ch <= '9')
        return true;

    // negative sign: only once, at the beginning
    if (ch == '-' && (m_flags & EFlags::AllowNegative))
    {
        return txt.IsEmpty();   // only allow if we are at start
    }
```

```
    // decimal point: only once, and allowed by flag
    if (ch == '.' && (m_flags & EFlags::AllowDecimal))
    {
        return !txt.Find('.');  // no existing decimal
    }

    return false;   // everything else is disallowed
}

// ---------------------------------------------------------------------------
// Helper: let the base class delete a character at caret position.
// ---------------------------------------------------------------------------
void CNumberEdit::HandleBackspace()
{
    // Let the default handler deal with it (caret positioning, etc.)
    CEdit::OnChar(VK_BACK, 0, 0);
}

// ---------------------------------------------------------------------------
// Insert the character at the current caret position and keep the caret
// after the new char.
// ---------------------------------------------------------------------------
void CNumberEdit::InsertChar(char ch)
{
    // Get selection range
    CHARRANGE cr;
    SendMessage(EM_EXGETSEL, 0, (LPARAM)&cr);

    // Build new string: replace selected text with the single character
    CString txt;
    GetWindowText(txt);
    txt.Delete(cr.cpMin, cr.cpMax - cr.cpMin);   // remove selection
    txt.Insert(cr.cpMin, ch);

    SetWindowText(txt);

    // Move caret to just after inserted char
    CHARRANGE newCr = {cr.cpMin + 1, cr.cpMin + 1};
    SendMessage(EM_EXSETSEL, 0, (LPARAM)&newCr);
}
```

---

3. How to use it

a) In your dialog’s header

```cpp
#include "NumberEdit.h"

class CMyDlg : public CDialogEx
{
public:
```

```
    // ...
protected:
    CNumberEdit m_editInt;   // integer only
    CNumberEdit m_editDec;   // decimal, allows negative numbers
};
```

b) In `DoDataExchange` (if you want DDX)

```cpp
void CMyDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialogEx::DoDataExchange(pDX);
    DDX_Control(pDX, IDC_EDIT_INT, m_editInt);
    DDX_Control(pDX, IDC_EDIT_DEC, m_editDec);
}
```

c) In the dialog's constructor

```cpp
CMyDlg::CMyDlg(CWnd* pParent /*=nullptr*/)
    : CDialogEx(IDD_MYDLG, pParent)
{
    // Nothing else needed – the control objects are already constructed.
}
```

Tip:
If you prefer to *subclass* an existing edit box that was created in the resource editor:

```cpp
// In your dialog's OnInitDialog()
CEdit* pOld = (CEdit*)GetDlgItem(IDC_EDIT_INT);
if (pOld)
{
    m_editInt.SubclassWindow(pOld->m_hWnd);   // take over the control
}
```

Now `IDC_EDIT_INT` will behave as a numeric-only edit box.
The same can be done for any other controls you wish to turn into `CNumberEdit`.

---

4. Customization

Feature	How to enable
Allow negative numbers	Pass `EFlags::AllowNegative` in the constructor: `m_editDec = CNumberEdit(CNumberEdit::EFlags::AllowNegative);`
Allow decimal point	Pass `EFlags::AllowDecimal`: `CNumberEdit m_editDec(EFlags::AllowDecimal | EFlags::AllowNegative);`

| Both negative and decimal | Combine flags with bitwise OR: `EFlags::AllowNegative | EFlags::AllowDecimal`. |

Feel free to extend the class (e.g., add range checks, formatting on lose-focus, etc.) – the core idea is the same: **filter `WM_CHAR`** and only accept characters that satisfy your rules.