June 8th, 2004, 03:59 PM #1

Andreas Masur
Moderator
Power Poster

Join Date: May 2000
Location: KY, USA
Posts: 18,652

C++ General: How to declare and use two-dimensional arrays?

Q: How can I define a dynamic two-dimensional array?

A: The following class shows one way of encapsulating a two-dimensional array:

Code:

```
template <class T>
class dynamic_2d_array
{
public:
  dynamic_2d_array(int row, int col) : m_row(row),
                                       m_col(col),
                                       m_data((row != 0 && col != 0) ? new T[row * col] : NULL){}

  dynamic_2d_array(const dynamic_2d_array& src) : m_row(src.m_row),
                                                  m_col(src.m_col),
                                                  m_data((src.m_row != 0 && src.m_col != 0) ? new T[src.m_row * s
  {
    for(int r = 0;r < m_row; ++r)
      for(int c = 0; c < m_col; ++c)
        (*this)[r][c] = src[r][c];
  }

  ~dynamic_2d_array()
  {
    if(m_data)
      delete []m_data;
  }

  inline T* operator[](int i) { return (m_data + (m_col * i)); }

  inline T const*const operator[](int i) const {return (m_data + (m_col * i)); }
```

```
private:
  dynamic_2d_array& operator=(const dynamic_2d_array&);
  const int m_row;
  const int m_col;
  T* m_data;
```

A different approach using the STL 'vector' class is shown in the following FAQ...

FAQ contributed by: [Axter]

Last edited by Andreas Masur; July 23rd, 2005 at 12:36 PM.

Quote

May 20th, 2005, 07:04 AM #2

NMTop40
Elite Member
Power Poster

Join Date: Oct 2000
Location: London, England
Posts: 4,773

Re: C++ General: How to declare and use two-dimensional arrays?

A: Here is another alternative approach. This uses a non-standard notation but is simple to use. I will call the class matrix.

Code:

```
template <typename T>
class Matrix
{
private:
   size_t m_nRows;
   size_t m_nCols;
   std::vector< T > m_vect;

public:
   Matrix( size_t nRows=0. size_t nCols=0, const T& t= T() )
    : m_nRows( nRows ), m_nCols (nCols), m_vect( nRows * nCols, t )
   {
   }

   size_t rows() const { return m_nRows; }
   size_t cols() const { return m_nCols; }

   T& operator()( size_t row, size_t col )
   {
        assert( row < m_nRows && col < m_nCols );
        return m_vect[ row * m_nCols + col ];
   }

   const T& operator()( size_t row, size_t col ) const
   {
        assert( row < m_nRows && col < m_nCols );
        return m_vect[ row * m_nCols + col ];
   }

   T* getRow( size_t  row )
   {
         assert( row <= m_nRows ); // we allow one past the end
                                   // this permits certain algorithms
```

That will give you reasonable functionality. You can do more with it (add a non-const iterator, give iterator more than FwdIterator traits).

Note: For matrices of numerical types, it may be better to use valarray instead of vector. Then a row could be returned as slicearray. I have never actually implemented one this way though.

Last edited by Andreas Masur; July 23rd, 2005 at 12:54 PM.

Quote

Post Reply

Posting Rules

You **may not** post new threads
You **may not** post replies
You **may not** post attachments
You **may not** edit your posts

BB code is **On**
Smilies are **On**
[IMG] code is **On**
HTML code is **On**

Forum Jump
CodeGuru Individual FAQs Go

All times are GMT -5. The time now is 09:52 PM.