

Các ví dụ về sử dụng cấu trúc dữ liệu đơn giản trong MFC

I. CString

II. CArray – CList - CStringArray

III. CSerialize

I. CString

CString là lớp chuỗi cơ bản trong MFC tương tự như char * trong C/C++ nhưng tự động quản lý kích thước và cấp phát vùng nhớ.

1. Các truy xuất cơ bản :

i. Gán :

```
CString mfcstring = "traditional string";
```

```
CString anotherstring = mfcstring;
```

Chú ý: nếu dịch chế độ unicode (_UNICODE) và muốn xử lý chuỗi unicode cần gán như sau :

```
CString mfcstring = _T("traditional string");
```

ii. Cộng:

```
CString s1 = "This ";
```

```
s1 += "is a ";
```

```
CString s2 = "test";
```

```
CString message = s1 + "big " + s2;
```

```
//Message se chua "This is a big test".
```

iii. So sánh ==, >, <..:

```
CString s1( "Tom" );
```

```
CString s2( "Jerry" );
```

```
if( s1 < s2 )
```

...

iv. Toán tử [] và ép kiểu sang char * hay LPCTSTR

```
CString s1 = "12345";
```

```
if(s1[0]=='1')
```

```
..... // s1[0]==1 se cho ket qua TRUE
```

```
CString kindOfFruit = "bananas";
```

```
int howmany = 25;
```

```
printf( "You have %d %s\n", howmany, (LPCTSTR)kindOfFruit );
```

```
// ep kindOfFruit than hkieu char * de su dung
```

!!!! Chú ý không được gán s[0] = '1' mà phải dùng hàm SetAt

```
s.SetAt(0,'1');
```

v. **Kích thước `GetLength()` và kiểm tra rỗng `IsEmpty()`**

```
CString s;
if(s.IsEmpty())
...// ketqua TRUE
s = "1234";
int len = s.GetLength();
//len = 4
```

vi. **Chữ hoa, chữ thường với `MakeUpper`, `MakeLower` (MSDN)**

2. **Format**

Hàm Format của `CString` tương tự như `sprintf` của C/C++, xem ví dụ minh họa sau:

```
CString mfcstring;
char *s = "sample";
int i = 12;
float f = "12.13";
mfcstring.Format("We have an integer(%d), a float(%f) and a %s.",i,f,s);
// ket qua se la "We have an integer(12), a float(12.13) and a sample."
```

3. **Tìm kiếm, trích, chèn, xóa ,thay thế**

i. **Tìm kiếm với `Find`**

```
int result;
CString s = "This is an example";
result = s.Find('i');
//kiem tu dau chuoit → result = 2
result = s.Find('i',3);
// kiem tu vi tri thu 3 → result = 5
result = s.Find("is");
// kiem tu dau chuoit → result = 2
result = s.Find("is",3);
// kiem tu vi tri thu 3 → result = 5
```

ii. **Trích với `Mid`, `Left`, `Right` (tham khảo trong MSDN)**

iii. **Chèn với `Insert` (tham khảo trong MSDN)**

iv. **Xóa với `Remove`, `Delete` (tham khảo trong MSDN)**

v. **Thay thế với `Replace` (tham khảo trong MSDN)**

II. Lớp CArray-CStringArray và lớp CList

Trước khi sử dụng hai lớp này, cần **#include <afxtempl.h>** trong **stdafx.h**

CArray và **CList** là hai kiểu dữ liệu tập hợp đơn giản trong MFC, cả hai đều hỗ trợ dạng template, có nghĩa, khi khai báo một mảng của một kiểu dữ liệu nhất định, ta chỉ cần khi báo kiểu đó vào mảng theo ví dụ như sau:

1. Ví dụ mảng/list kiểu cơ bản

```
CArray<int,int> intarray;
intarray.Add(1);
intarray[0] = 12;
cout<<intarray[0]<<endl;
CList<int,int> ilist;
ilist.AddTail(1);
```

2. Ví dụ mảng/list kiểu lớp:

```
class CPerson
{
protected: int data;
public:
    void abc(){cout<<data<<endl;};
};
Cperson person;
CArray<CPerson, CPerson&> parray;
parray.Add(person);

CList<CPerson, CPerson&> plist;
plist.AddTail(person);
```

3. Các thao tác trên CArray:

- **SetSize(int size)** : thiết lập kích thước ban đầu
- **GetSize()** : lấy kích thước mảng
- **RemoveAll()** : Xóa hết tất cả phần tử trong mảng.

!!! **Chú ý** : khi mảng chứa các con trỏ, nếu gọi **RemoveAll()** thì các bộ nhớ do các con trỏ chiếm giữ sẽ không được hủy, do đó phải hủy từng con trỏ trong mảng trước khi gọi hàm này.

- **SetAt(int i, data)** gán giá trị data cho phần tử thứ i, **GetAt(int i)** trả về phần tử thứ i, còn **ElementAt(int i)** và **[]** trả về tham chiếu tới phần tử thứ i, ví dụ :

```
int dat = array.GetAt(i);
dat = 5; // phần tử thứ i trong mảng không thay đổi gì
int& ref = array.ElementAt(i);
ref = 5; // kết quả phần tử thứ I trong mảng thay đổi theo
```

- Các hàm **Add()**, **Append()**, **InsertAt()**, **RemoveAt()** tham khảo trong MSDN.

4. Các thao tác trên CList:

Do việc truy xuất trên list không thể tuần tự như với mảng, MFC đề ra ý tưởng con trỏ **position**, ta bắt đầu bằng việc cho con trỏ ngay đầu danh sách hoặc cuối danh sách. Với con trỏ, chúng ta có thể cho con trỏ di chuyển tiếp sau hoặc trở về trước, và có thể truy xuất phần tử của dữ liệu tại vị trí con trỏ.

- **GetCount()** lấy kích thước danh sách
- **POSITION** là kiểu dữ liệu con trỏ
- **GetHeadPosition()** và **GetTailPosition()** là hai hàm thiết lập vị trí ban đầu của con trỏ
- **DATA GetNext(POSITION &pos)** và **DATA GetPrev(POSITION &pos)** trả về giá trị dữ liệu tại con trỏ và di chuyển con trỏ về trước hay sau.
- **GetAt(POSITION pos)** và **SetAt(POSITION pos, DATA)** là hai hàm truy xuất dữ liệu tại vị trí con trỏ
- **InsertBefore(POSITION pos, DATA)** và **InsertAfter(POSITION pos, DATA)** chèn dữ liệu trước hay sau con trỏ
- **RemoveAt(POSITION pos)** xóa phần tử tại vị trí con trỏ
- **POSITION FindIndex(int index)** → lấy con trỏ của phần tử thứ **index**.
- **Find, IsEmpty, RemoveAll.v.v....** tham khảo trong MSDN

!!! **Chú ý** : Tương tự như mảng, khi danh sách chứa các con trỏ, nếu gọi **RemoveAll()** thì các bộ nhớ do các con trỏ chiếm giữ sẽ không được hủy, do đó phải hủy từng con trỏ trong danh sách trước khi gọi hàm này.

Ví dụ sau minh họa truy xuất phần tử thứ 5 của danh sách, và truy xuất lần lượt từ đầu danh sách đến cuối danh sách:

```
CList<int, int> ilist;
// thêm các phần tử
```

```

for(int i=0;i<10;i++)
    ilist.AddTail(i+1);
// truy xuất phần tử thứ 5
POSITION pos ;
pos = ilist.FindIndex(4);
cout << ilist.GetNext(pos)<<endl;
// truy xuất từ đầu đến cuối
pos = ilist.GetHeadPosition();
while(pos)
{
    // lấy ra giá trị và di chuyển con trỏ tiếp
    cout << ilist.GetNext(pos)<<endl;
}
// xóa tất cả các phần tử
ilist.RemoveAll();

```

5. Các thao tác trên CStringArray:

Bản thân **CStringArray** chính là **CArray** chứa các **CString** nên các thao tác cũng tương tự như **CArray**. Tham khảo trong MSDN.

III. Tạo các lớp Serializable

MFC hỗ trợ việc lưu trữ và đọc các đối tượng theo mô hình Serialize. Một lớp **Serializable** là một lớp có thể tự lưu dữ liệu mình vào một tập tin nhị phân và tự đọc lên từ tập tin đó. Ví dụ sau minh họa cách tạo một lớp Serializable

Trong tập tin Person.h

```

class CPerson : public CObject
{
public:
    DECLARE_SERIAL( CPerson )
    // Đăng ký serializable
    CString m_name;
    WORD m_number;
    void Serialize( CArchive& archive );
    // Hàm truy xuất tại đây
};

```

Trong tập tin Person.cpp

```
IMPLEMENT_SERIAL( CPerson, CObject, 1 )
// CObject là lớp cha
void CPerson::Serialize( CArchive& archive )
{
    // Gọi Serialize của lớp cha trước
    //
    CObject::Serialize( archive );

    // Lưu trữ
    if( archive.IsStoring() )
        archive << m_name << m_number;
    else // đọc lên
        archive >> m_name >> m_number;
}
```

Sau khi tạo một lớp **serialize**, chúng ta có thể đọc ghi một lớp một cách dễ dàng bằng việc gọi các **operator >> <<** như sau:

```
CPerson person;
.....
CFile f;
f.Open("abc.bin",CFile::modeWrite | CFile::modeCreate);
// ngược lại đọc là CFile::modeRead
CArchive ar (&f, CArchive::store);
// ngược lại đọc là CArchive::load
ar << person;
// ngược lại đọc là ar >> person;
ar.Close();
f.Close();
```

Chú ý: Lớp **Serializable** phải thừa kế từ **CObject**. Đối với các lớp phức tạp, ta có thể áp dụng **archive** cho tất cả các thành viên của lớp miễn là các thành viên đó thuộc các kiểu cơ bản hoặc các lớp là **serializable**. Ví dụ sau minh họa serialize lớp **CPersonList**:

```
public class CPersonList : public CObject
{
public:
    DECLARE_SERIAL( CPersonList )
```

```
    CList<CPerson, CPerson &> m_list;
    CString m_strListName;
    ....

    void Serialize( CArchive& archive );
}

IMPLEMENT_SERIAL( CPersonList, CObject, 1 )

void CPersonList::Serialize( CArchive& archive )
{
    CObject::Serialize( archive );

    if( archive.IsStoring() )
    {
        // m_strListName la CString nen Serializable
        // GetCount tra ve int cung serializable
        archive << m_strListName << m_list.GetCount();
        POSITION pos = m_list.GetHeadPosition();
        while(pos)
        {
            // vi CPerson da la lop serializable
            archive << m_list.GetNext(pos);
        }
    }
    else
    {
        int num;
        m_list.RemoveAll();
        // m_strListName la CString nen Serializable
        // num la int cung serializable
        archive >> m_strListName >> num;
    }
}
```

```
        for(int i=0;i<num;i++)
        {
            CPerson person;
            // vi CPerson da la lop serializable
            archive >> person;
            m_list.Append(person);
        }
    }
}
```

Khi đọc hai ghi **CPersonList** vào tập tin ta chỉ việc làm như sau :

```
CPersonList list;
.....
CFile f;
f.Open("abc.bin",CFile::modeWrite | CFile::modeCreate);
// ngược lại đọc là CFile::modeRead
CArchive ar (&f, CArchive::store);
// ngược lại đọc là CArchive::load
ar << list;
// ngược lại đọc là ar >> person;
ar.Close();
f.Close();
```