C++ 코딩 테크닉
STL 맵을 사용한 환경 변수 관리
신영진, pop@jiniya.net http://www.jiniya.net

처음 상용 프로그램을 개발하는 신입 프로그래머들은 두 가지 사실에 굉장히 놀란다. 과도하리만큼 많은 수의 if문과 지나칠 정도로 다양한 옵션이 그것이다. 이 두 가지 모두 학교에서 프로그램을 작성할 때에는 쉽게 배울 수 없는 것들이다. 많은 if문은 그만큼 여러 가지 경우를 대비한다는 것을, 많은 수의 환경 변수는 그만큼 폭넓은 고객의 요구를 반영한다는 의미로 풀이할 수 있다. 하지만 이러한 것들이 잘 관리되면 좋겠지만 실제 현업에서는 그렇지 못한 경우가 많다. 다년간 많은 사람들의 손을 거쳐간 코드는 왜 달렸는지도 모를 if문과 예측조차 되지 않는 이름의 전역변수투성이다. 유지보수 하는 사람에게는 암호와도 같은 것이다. 이러한 환경 변수를 좀 더 체계적으로 관리할 수 있는 방법에 대해서 알아보자.

<리스트 1>에는 환경 변수를 관리해주는 CVars 클래스가 나와있다. 이 클래스의 기능은 단순하다. 환경 변수를 등록하고, 변경하고, 조회하는 기능이 전부다. 모든 환경 변수는 이름을 통해서 구분되기 때문에 동일한 이름의 환경 변수를 등록할 순 없다. 환경 변수로 등록할 수 있는 타입은 템플릿 멤버로 넘어온 T다. 일반적으로 환경 변수의 경우 문자열, 정수, 실수, 이미지 등으로 그 범위가 넓다. 따라서 T는 이러한 모든 자료를 다 저장하고 복원할 수 있는 구조가 되어야 할 것이다. <리스트 2>에 그러한 용도에 사용될 수 있는 CVarItem 클래스의 원형이 나와있다.

**리스트 1 환경 변수 저장 클래스**

```
// 환경 변수 저장 클래스
template <class T>
class CVars
{
private:
   typedef std::map<std::string, T>                VarMap;
   typedef typename std::map<std::string, T>::iterator VarMIt;

   VarMap m_vars;  // 환경 변수를 저장할 맵
   T m_nullItem;   // 변수를 찾지 못한 경우 리턴할 널값

public:
   // 환경 변수 등록 함수
   BOOL Register(LPCTSTR name, T item)
   {
      VarMIt it = m_vars.find(name);
      if(it != m_vars.end())
         return FALSE;
```

```
        m_vars.insert(make_pair(name, item));
        return TRUE;
    }

    // 환경 변수 값 변경 함수
    BOOL Set(LPCTSTR name, T item)
    {
        VarMIt it = m_vars.find(name);
        if(it == m_vars.end())
            return FALSE;

        it->second = item;
    }

    // 환경 변수 값 조회 함수
    const T &Get(LPCTSTR name)
    {
        VarMIt it = m_vars.find(name);
        if(it == m_vars.end())
            return m_nullItem;

        return it->second;
    }
};
```

우리와 같이 한 컨테이너에 여러 종류의 타입의 데이터를 넣고 싶은 경우에 사용할 수 있는 방법엔 두 가지가 있다. 하나는 <리스트 2>와 같이 union을 사용해서 저장하고 싶은 데이터를 모두 가지고 있는 클래스를 만드는 방법이고, 다른 한 가지는 CVarItem을 가상 클래스로 만들어서 여기서 새로운 타입에 대응하는 클래스를 상속받아서 만드는 방법이다. 저장하고 싶은 데이터가 간단하고 범위가 넓지 않다면 union을 사용하는 방식이 속도적인 측면에서 이득이다.

**리스트 2 CVarItem 클래스**

```
class CVarItem
{
private:
    enum
    {
        VT_EMPTY,
        VT_INT,
        VT_UINT,
        VT_STRING,
        VT_COLORREF,
        VT_DOUBLE
    };

    int    m_type;
    union
    {
```

```
        int m_iVal;
        std::string *m_strVal;
        COLORREF m_clrVal;
        UINT m_uiVal;
        double m_dblVal;
    };

public:
    // 생성자들
    CVarItem();
    CVarItem(const CVarItem &r);
CVarItem(LPCTSTR strVal);
    CVarItem(int iVal);

    // 대입 연산자들
    CVarItem &operator =(const CVarItem &r);
    CVarItem &operator =(int iVal);
    CVarItem &operator =(UINT uiVal);
    CVarItem &operator =(COLORREF clrVal);

    // 변환 함수들
    operator int() const;
    operator COLORREF() const;
};
```

CVarItem 같은 클래스를 만드는 것이 어려운 작업은 아니지만 굉장히 귀찮고 지루한 일이다. 사용하고 싶은 데이터의 범위가 일반적인 타입이 전부라면 VARIANT를 포장한 _variant_t 클래스를 사용하는 것이 좋다. 이 경우 단점이라면 문자열을 얻어올 때 _bstr_t로 한번 변환해 주어야 한다는 점이다. <리스트 3>에 _variant_t를 사용한 샘플 코드가 나와있다.

**리스트 3 CVars 클래스를 사용한 예제 코드**

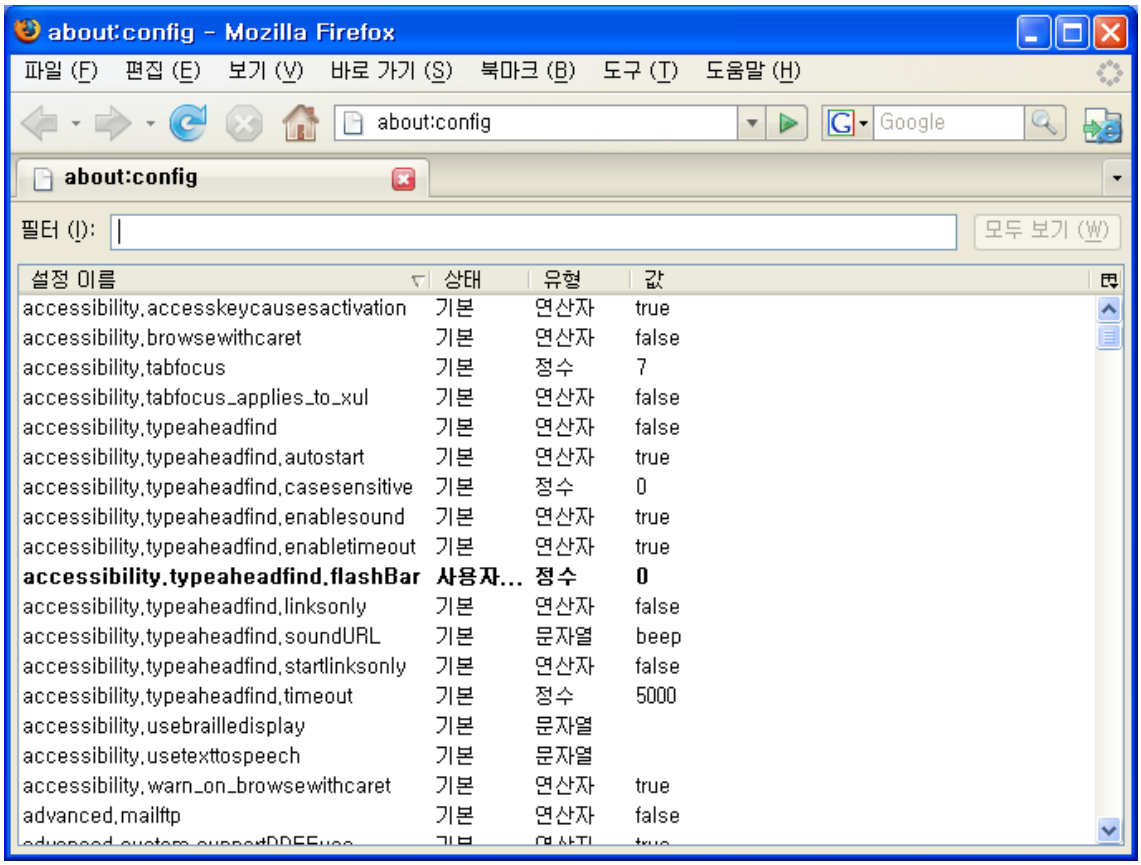
```
CVars<_variant_t> g_vars;

int _tmain(int argc, _TCHAR* argv[])
{
    // 환경 변수를 등록한다
    g_vars.Register("path", "c:\\program files");
    g_vars.Register("shutdown time", 30);
    g_vars.Register("pi", 3.141592);

    // 환경 변수 값을 변경한다.
    g_vars.Set("path", "d:\\comma");

    // 환경 변수 값을 조회한다.
    _bstr_t bstr = g_vars.Get("path");

    double pi = g_vars.Get("pi");
    printf("%f %d %s\n", pi, (int) g_vars.Get("shutdown time"), (const char *)bstr);
    return 0;
}
```




**화면 1 파이어폭스 2.0에 사용되는 환경 변수들**

<화면 1>은 파이어폭스 2.0에 사용된 환경 변수 목록이다. 주소 창에 about:config 이라고 입력하면 이 화면이 나온다. 앞서 설명한 CVars 클래스를 사용한다면 저렇게 많은 환경 변수를 관리하는 것도 어렵지 않다. 아마 파이어폭스의 환경 변수 목록을 본다면 전역 변수보다는 CVars 클래스의 접근법이 훨씬 더 효율적임을 알 수 있을 것이다.

CVars 클래스에 한 가지 아쉬운 점이 있다면 환경 변수의 설명을 추가할 수 없다는 점이다. 물론 이름이 변수명 보다는 훨씬 더 자세한 정보를 제공하지만, 이 방법도 변수가 많아지면 코드를 읽는 다음 사람에게는 전역 변수나 비슷하다. Register 함수를 수정해서 이름과 함께 자세한 설명을 추가할 수 있도록 고쳐보자.