关于 C++指针使用的重要附加内容

姚尧 中山大学

注：

文中出现的 TT 为任意 C++标准数值类型，可以为：

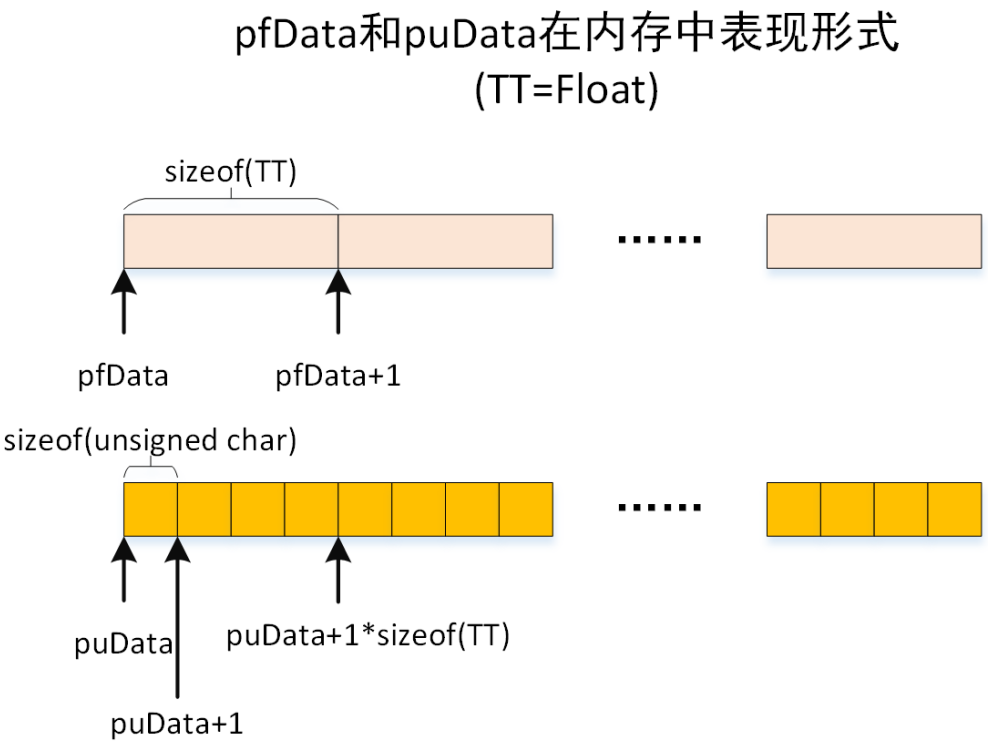
C++标准数值类型 TT	其他名称	位数	字节数=位数/8 sizeof(TT) =	备注
unsigned char	BYTE byte GDT_Byte	8	1	32 位/64 位程序
char	signed char	8	1	32 位/64 位程序
unsigned short	unsigned int16 uint16 UINT16 GDT_UInt16	16	2	32 位/64 位程序
short	int16 INT16 GDT_INT16	16	2	32 位/64 位程序
unsigned int	UINT32 GDT_UInt32	32	4	32 位/64 位程序
int	INT32 GDT_INT32	32	4	32 位/64 位程序
float	float32 FLOAT32 GDT_Float32	32	4	32 位/64 位程序
double	float64 FLOAT64 GDT_Float64	64	8	32 位/64 位程序
long long	long long int	64	8	64 位程序
long double		128	16	64 位程序

- **开辟 TT 类型大小为 N 内存空间：**

  方法 1：TT* pfData = new TT[N];

  方法 2：unsigned char* puData = new unsigned char[N*sizeof(TT)];

  此时，pfData 和 puData 所开辟的内容空间是等同的，但是两者直接移动移动 1 位是不一致的。在内存中表现方式如下：



- **指针移动到位置 i 并取出 TT 类型的数据**

  方法 1：pfData[i]

  方法 2：*(pfData + i)

  方法 3：*((TT*)puData + i)

  方法 4：((TT*)puData)[i]

  方法 5：*(TT*)(puData + i*sizeof(TT))

  其中，方法 5 是最重要的方法：

  第一步，将指针位移到对应的位置：unsigned char* p = puData + i*sizeof(TT);

  第二步，更改指针数据类型：TT* p1 = (TT*)p;

  第三步，取值：TT val = *p1;

- **将 TT 类型的数据 val 写入 TT 类型数组 的 i 位置**

  方法 1（正确）：pfData[i] = val; 或 *(pfData+i) = val;

  方法 2（正确，不常用）：memcpy(pfData+1, &val, 1);

  方法 3（正确，常用）：memcpy(puData+i*sizeof(TT), &val, sizeof(TT));

  方法 4（正确，常用）：*((TT*)puData + i) = val;

  ~~方法 5~~（错误，写入空间不够）：*(puData + i*sizeof(TT)) = val;

方法 3 和方法 4 是在大量数据处理中最常用的 memcpy 方法，请理解和牢记。

- **用一维 byte 数组表示 宽度 w*高度 h*波段数 b 的多维 TT 类型数据**

  定义：unsigned char* pData = new unsigned char[w*h*b*sizeof(TT)];

  置零：

  方法 1：memset(pData, 0, w*h*b*sizeof(TT));

  方法 2：

  BSQ 格式（GDAL）：

```
for(j=0; j<h; j++)
{
    for(i=0; i<w; i++)
    {
        for(k=0; k<b; k++)
        {
            TT val = 0;
            *((TT*)pData + k*w*h + i*h + j) = val;
        }
    }
}
```

  BIP 格式（openCV）：

```
for(j=0; j<h; j++)
{
    for(i=0; i<w; i++)
    {
        for(k=0; k<b; k++)
        {
            TT val = 0;
            *((TT*)pData + i*w*b + j*b + k) = val;
        }
    }
}
```

  取值：

  BSQ 格式：TT val = *((TT*)pData + k*w*h + i*h + j);

  BIP 格式：TT val = *((TT*)pData + i*w*b + j*b + k);

- **遇到未知类型的遥感影像数据处理方法（一般格式，仅供参考）**

  假设存在 IMGPROC 类，类中已有一个 GDAL 对象 mpoDataset。

  第一步，在 IMGPROC 类中定义模板函数 process（保护或私有），实现有关数据处理算法：

template<class TT> bool IMGPROC::process();

第二步，定义一个普通函数 standardProcess（公有），供外部调用。函数内容如下：

```
bool IMGPROC::standardProcess()
{
    GDALDataType dt = mpoDataset->GetRasterBand(1)->GetRasterDataType();

    bool bRlt = false;
    switch(mgDataType)
    {
    case GDT_Byte:
        bRlt = process<unsigned char>();
        break;
    case GDT_UInt16:
        bRlt = process<unsigned short>();
        break;
    case GDT_Int16:
        bRlt = process<short>();
        break;
    case GDT_UInt32:
        bRlt = process<unsigned int>();
        break;
    case GDT_Int32:
        bRlt = process<int>();
        break;
    case GDT_Float32:
        bRlt = process<float>();
        break;
    case GDT_Float64:
        bRlt = process<double>();
        break;
    default:
        cout<<"unknown data type!"<<endl;
        return false;
    }

    return bRlt;

}
```

这样，外部通过定义 IMGPROC 对象 obj，然后 obj.standardProcess()就可以处理任意类型的遥感影像数据了。