Variables and Arrays

- C++ has basic data types such as short, int, float, char, double etc.

- The statements
  ```
  int a;
  float b[10];
  double c[5][5];
  ```
  define a single integer a, an one-dimensional array of floats b[0] .. b[9] and a two-dimensional array of doubles c[0][0] .. c[4][4] (All array indices start at 0.)

- Both a, b and c are implicitly allocated and will be deallocated when the function containing them exits. Their sizes are fixed at compile time.

Dynamic Memory Allocation

- Dynamic allocation allows the creation of arrays whose size is determined at runtime (e.g. loading an image whose size can vary).

- It is one of the key to writing memory efficient programs.

- It is, arguably, also the biggest source of problems and crashes in most C++ code. Most of the problems are caused by:
  1. Accessing/Deleting arrays/objects before they are allocated and initialized.
  2. Accessing/Deleting arrays/objects after they have been already deleted
  3. Neglecting to delete arrays/objects
Allocating/De-allocating Pointers

- Memory allocation is performed by the `new` command e.g.
  `int* a = new int;`
- `a` is NOT an integer. It is a pointer which contains a memory location. To access the value stored at the memory location pointed by `a` use the de-referencing operator `*` e.g.
  `*a = 1`  //This sets the value of the integer whose location is stored in `a` to 1
- When `a` is no longer needed the memory can be released using the `delete` command e.g.
  `delete a;`

Allocating/De-allocating Arrays of Pointers

- As before allocation is performed by the `new` command e.g.
  `int* a = new int[10];`  // Allocate an array of 10 integers
- `a` is a pointer which contains the memory location of the first element of the array. To access the values stored at the memory locations use:
  `a[0] = 1; a[3] = 2;` etc.
- When `a` is no longer needed the memory can be released using the `delete []` command e.g.
  `delete [] a;`
- Do not use the `delete` operator to delete arrays, it causes lots of problems use `delete []` !!!

Class Design

Classes consists of:

1. Data members -- just like C structures
2. Methods – special functions which are embedded in the class

Members and methods can be declared as public which makes them accessible from outside the class or protected/private which makes them inaccessible from outside the class methods.

Inheritance

- Previous trivial example showed how classes can be extended by deriving new classes from them
- Derived or child classes can also override standard behavior by redefining member functions.
- Can have abstract base classes which simply define an interface and leave the implementation to derived classes.
- The combination of multiple layers of derived classes leads to class hierarchies.

VTK and Macros

- VTK uses certain macros to help with the parsing of the C++ header files and the creation of wrapper code for TCL/Java/Python.
- The most common are:
  `vtkSetMacro(member,type);`  `vtkGetMacro(member,type);`
  e.g. `vtkSetMacro(Length,float)` results in
  ```
  void SetLength(float f) { Length=f; }
  ```
  and `vtkGetMacro(Length,float)` results in
  ```
  float GetLength() { return Length; }
  ```
  `vtkGetObjectMacro(object,type)` and `vtkGetObjectMacro(object,type)` are used to access/modify pointer objects derived from `vtkObject`
Image to Image Filters

Two key filter methods:
- Execute Information()
- Execute()

Talk Outline
- A Review of last week's lecture
- Adding Image to Image Filters using the vtkSimpleImageToImageFilter
- Adding Multiple Image to Image Filters using the vtkpxMultipleSimpleImageToImageFilter
- Adding Complex Image Filters using the multithreaded vtkImageToImageFilter

The vtkSimpleImageToImageFilter
- The execute method by default calls a function called SimpleExecute which has the specification
  
  ```c++
  virtual void SimpleExecute(vtkImageData* input, vtkImageData* output);```
- Input is the input to the filter set with the SetInput() method
- Output is the output of the filter
- If output image has the same size as input image then the output image is correctly allocated.
- All the user needs to do is to define the SimpleExecute() method

The vtkpxSimpleThreshold Filter
- Simple method which takes an input image and outputs
  - 1 if lower threshold<=intensity<=upper threshold
  - 0 otherwise
- Two new data members (as opposed to those inherited)
  - LowerThreshold
  - UpperThreshold
- Three new methods
  - New()
  - Constructor
  - SimpleExecute()
**vtkpxSimpleThreshold.h**

- Public Interface of class is fairly standard
- Typical New/Type Macro Setup
- Use of `vtkSetMacro` and `vtkGetMacro` for accessing the thresholds

```cpp
#include "vtkObjectFactory.h"
#include "vtkpxSimpleImageThreshold.h"
nvtkpxSimpleImageThreshold* vtkpxSimpleImageThreshold::New(){
  // First try to create the object from the vtkObjectFactory
  vtkObject* ret = vtkObjectFactory::CreateInstance("vtkpxSimpleImageThreshold");
  if (ret)
    return (vtkpxSimpleImageThreshold*)ret;
  // If the factory was unable to create the object, then create it here.
  return new vtkpxSimpleImageThreshold;
}
```

**vtkpxSimpleThreshold.cpp**

```cpp
void vtkpxSimpleImageThreshold::SimpleExecute(vtkImageData* input, vtkImageData* output){
  if (input==NULL)    {
    vtkErrorMacro ("Bad Input to vtkpxSimpleImageThreshold");
    return;
  }
  int dim[3]; input->GetDimensions(dim);
  int numvox=dim[0]*dim[1]*dim[2],
  int numcomp=input->GetNumberOfScalarComponents();
  vtkDataArray* inarray=input->GetPointData()->GetScalars();
  vtkDataArray* outarray=output->GetPointData()->GetScalars();
  for (int component=0;component<numcomp; component++)    {
    for (int i=0;i<numvox; i++) {
      float out=0.0;
      float v=inarray->GetComponent(i,component);
      if (v>=this->LowerThreshold && v<=this->UpperThreshold)
        out=1.0;
      outarray->SetComponent(i,component,out);
    }
  }
}
```

**vtkpxImageExtract.h**

- The `vtkSetClampMacro()` is used to restrict the values set for CurrentPlane in the range 0 to 2
- The rest of the code is fairly standard...

```cpp
#include "vtkSimpleImageToImageFilter.h"
class vtkpxImageExtract : public vtkSimpleImageToImageFilter {
  ...
  // Description:  // CurrentPlane 0=YZ, 1=XZ, 2=XY, slice= sliceno
  vtkSetClampMacro(CurrentPlane,int,0,2);
  vtkGetMacro(CurrentPlane,int);
  // Description:  // Set Slice Number 0 --Dim-1
  vtkSetMacro(SliceNo,int);
  vtkGetMacro(SliceNo,int);
  // Description:  select frame
  vtkSetMacro(Frame,int);
  vtkGetMacro(Frame,int);
  protected:
  vtkpxImageExtract();
  virtual void SimpleExecute(vtkImageData* input, vtkImageData* output);
  virtual void ExecuteInformation();
  ...
};
```
vtkpxImageExtract.cpp - 1

• Constructor is fairly trivial. It sets the default values
```cpp
vtkpxImageExtract::vtkpxImageExtract(){
this->CurrentPlane=2;
this->SliceNo=0;
this->Frame=0;
}
```

vtkpxImageExtract.cpp - 2

• The ExecuteInformation Method is typical of this type
  – First call the parent class ExecuteInformation
  – Then check if the input is defined if not set some standard defaults and exit
  – Then set the spacing/origin which are the same regardless of the extracted slice
```cpp
void vtkpxImageExtract::ExecuteInformation()
```

vtkpxImageExtract.cpp - 3

• (continued)
  – Then get the dimensions and depending on the slice type set the output extent
    – Note that the z-extent is 0..0 as the output image is 2D
```cpp
int wholeExtent[6], dims[3];
input->GetDimensions(dims);
wholeExtent[0]=0; wholeExtent[2]=0; wholeExtent[4]=0;
wholeExtent[5]=0;
switch (this->CurrentPlane )
{
  case 2: // XY Plane
    wholeExtent[1] = dims[0]-1;
    wholeExtent[3] = dims[1]-1;
    break;
  case 1: // XZ Plane
    wholeExtent[1] = dims[0]-1;
    break;
  case 0: // YZ Plane
    wholeExtent[1] = dims[1]-1;
    break;
}
output->SetWholeExtent (wholeExtent);
output->SetScalarType(input->GetScalarType());
output->SetNumberOfScalarComponents(1);
```

vtkpxImageExtract.cpp - 4

• The SimpleExecute() method is nothing special it simply copies the appropriate voxels into the output image
```cpp
void vtkpxImageExtract::SimpleExecute(vtkImageData* input,
  vtkImageData* output) {
  int dim[3];
  input->GetDimensions(dim);
  float ori[3];
  input->GetOrigin(ori);
  float sp[3];
  input->GetSpacing(sp);
  vtkDataArray* inp=input->GetPointData()->GetScalars();
  int dty=inp->GetDataType();
  output->SetSpacing(sp);
  output->SetScalarType(dty);
  output->SetNumberOfScalarComponents(1);
  output->SetOrigin(0.0,0.0,0.0);
  …..
```

vtkpxImageExtract.cpp - 5

```cpp
void vtkpxImageExtract::SimpleExecute(vtkImageData* input,
  vtkImageData* output) {
….
switch(this->CurrentPlane )
{
  case 2:
    output->SetDimensions(dim[0],dim[1],1);
    output->SetWholeExtent (0,dim[0]-1,0,dim[1]-1,0,0);
    output->AllocateScalars();
    inp->GetPointData()->GetScalars();
    int offset=dim[0]*dim[1]*this->SliceNo;
    for (int i=0;i<dim[0];i++)
      for (int j=0;j<dim[1];j++)
        output->GetPointData()->GetScalars();
        for (int k=0;k<dim[2];k++)
          output->GetComponent(i+j*dim[0],k+offset,0)=input->GetPointData()->GetScalars();
```

Talk Outline

• A Review of last week’s lecture
• Adding Image to Image Filters using the `vtkSimpleImageToImageFilter`
• Adding Multiple Image to Image Filters using the `vtkpxMultipleSimpleImageToImageFilter`
• Adding Complex Image Filters using the multithreaded `vtkImageToImageFilter`
vtkpxMultipleSimpleImageToImageFilter

- This is an abstract parent class derived from vtkSimpleImageToImageFilter that adds the following functions:
  - virtual void SetInput(int num, vtkImageData* input)
  - virtual void AddInput(vtkImageData* input)
  - vtkImageData* GetInput(int num)

- These allow for the specification of multiple inputs e.g. in the case of a filter that outputs the average of its inputs.
- This is custom code (not part of the original VTK distribution) and can be found in the examples available at the web page.

Implementing an Abstract Class

- vtkpxMultipleSimpleImageToImageFilter is an abstract class – it is never meant to be instantiated.
- There is no New() function.
- In CMakelist.txt there is an additional entry marking this as abstract:
  ```cmake
  ABSTRACT_FILES(vtkpxMultipleSimpleImageToImageFilter)
  ```

vtkpxAverageImages

- This class has functionality for computing the average of multiple images.
- Typical code:
  ```cpp
tkpxAverageImages* ave = vtkxAverageImages::New();
  ave->AddInput(img1);
  ave->AddInput(img2);
  ave->AddInput(img3);
  ave->AddInput(img4);
  ave->Update();
  ```
- The actual code in the examples has facilities for computing Median instead of Mean and other tweaks not shown here.

vtkpxAverageImages.h

- Only non-standard stuff shown (boiler plate stuff excluded).
- Only non-standard stuff shown (boiler plate stuff excluded also most vtkGetMacro()).

```cpp
#include "vtkpxMultipleSimpleImageToImageFilter.h"

class vtkxAverageImages : public vtkpxMultipleSimpleImageToImageFilter
{
public:
  // Description: Compute StandardDeviation -- to be obtained from StandardDeviationOutput
  // Description: Get the Output Standard Deviation
  void SetIgnoreFlag(int); // Ignore Flag -- if set to 1 if any one image has a pixel value = Ignore Value
  void SetIgnoreValue(float); // Ignore FlagOn() and IgnoreFlagOff() etc
  void SetIgnoreValueOutput(float); // Which are equivalent to SetIgnoreFlag(1) and SetIgnoreFlag(0)
  vtkBooleanMacro(IgnoreFlag, int);
  vtkBooleanMacro(IgnoreFlagOn(), int);
  vtkBooleanMacro(IgnoreFlagOff(), int);
  vtkSetMacro(IgnoreValue, float);
  vtkSetMacro(IgnoreValueOutput, float);
  vtkGetMacro(OutputStandardDeviation, vtkImageData);
};
```

vtkpxAverageImages.cpp

- Constructor and Destructor Shown. Note use of =NULL check.

```cpp
tkpxAverageImages::vtkxAverageImages()
{
  this->IgnoreFlag=0;
  this->IgnoreValue=-32767.0;
  this->IgnoreValueOutput=0.0;
  this->ComputeStandardDeviation=0;
  this->OutputStandardDeviation=NULL;
}
```
Talk Outline

- A Review of last week’s lecture
- Adding Image to Image Filters using the vtkSimpleImageToImageFilter
- Adding Multiple Image to Image Filters using the vtkMultipleSimpleImageToImageFilter
- Adding Complex Image Filters using the multithreaded vtkImageToImageFilter

Case Studies

- Choices
  1. Point-based registration (ICP)
  2. Image-intensity based registration
  3. Segmentation using snakes

  - The idea would be to show how the design process will work rather than focusing too much on the underlying code
  - Highlight key VTK tools for making the implementation of these easier

vtkImageToImageFilter

- Most internal image to image filters are based on vtkImageToImageFilter
- These uses more advanced techniques such as:
  - Multithreading for processing the image in pieces and taking advantage of multiple processors to do this in parallel
  - Direct access to the underlying pointers (i.e. no GetComponent()/SetComponent()) stuff and templated functions for fast implementations.
- Key Entry point is now the ThreadedExecute() function
- Will not show code here look at VTK source code for lots of examples.

Talk Outline

- A Review of last week’s lecture
- Adding Image to Image Filters using the vtkSimpleImageToImageFilter
- Adding Multiple Image to Image Filters using the vtkMultipleSimpleImageToImageFilter
- Adding Complex Image Filters using the multithreaded vtkImageToImageFilter

Case Studies

- Choices
  1. Point-based registration (ICP)
  2. Image-intensity based registration
  3. Segmentation using snakes

  - The idea would be to show how the design process will work rather than focusing too much on the underlying code
  - Highlight key VTK tools for making the implementation of these easier
Homework

• Download example1cpp.zip from class web-page this has been updated.

• Run CMAKE on it and compile it.
  – Warning Unix setup will only work on our machines with or in /usr/local/vtk.
  – Windows setup needs a workdirphet 4.5 or 2 and cmake 1.2. Will work with later cmake
    in 1.2 compatibility mode, set version 1.2.

• Next seminar next week.