∼

Starting on UNIX

Scientific Computing Winter 2016/2017

Lecture 5

Jürgen Fuhrmann

juergen.fuhrmann@wias-berlin.de

With material from from `http://www.cplusplus.com/` and from "Introduction to High-Performance Scientific Computing" by Victor Eijkhout (`http://pages.tacc.utexas.edu/~eijkhout/istc/istc.html`)

made wit pandoc

Recap from last time

Inheritance

- Classes in C++ can be extended, creating new classes which retain characteristics of the base class.
- The *derived class* inherits the members of the *base class*, on top of which it can add its own members.

```
class vector2d
{
private:
    double *data;
    vector2d<int> shape;
    int size
public:
    double & operator(int i, int j);
    vector2d(int nrow, ncol);
    ~vector2d();template <t
}

class matrix: public vector2d
{
   public:
    apply(const vector1d& u, vector1d &v);
    solve(vector1d&u, const vector1d&rhs);
}
```

- All operations which can be performed with instances of `vector2d` can be performed with instances of `matrix` as well
- In addition, `matrix` has methods for linear system solution and matrix-vector multiplication

Smart pointers

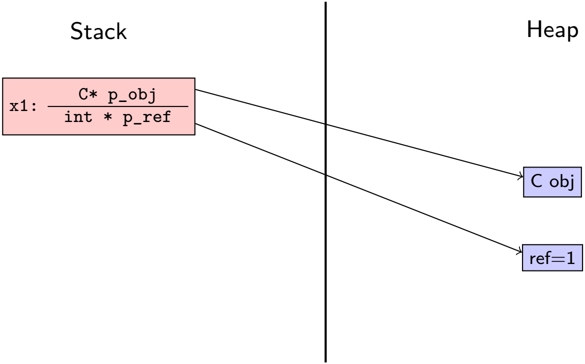
... with a little help from Timo Streckenbach from WIAS who introduced smart pointers into our simulation code.

- Automatic book-keeping of pointers to objects in memory.
- Instead of the meory addres of an object aka. pointer, a structure is passed around *by value* which holds the memory address and a pointer to a *reference count* object. It delegates the member access operator `->` and the address resolution operator `*` to the pointer it contains.
- Each assignment of a smart pointer increases this reference count.
- Each destructor invocation from a copy of the smart pointer structure decreses the reference count.
- If the reference count reaches zero, the memory is freed.
- `std::shared_ptr` is part of the C++11 standard

Smart pointer schematic

(this is one possibe way to implement it)

```
class C;
```

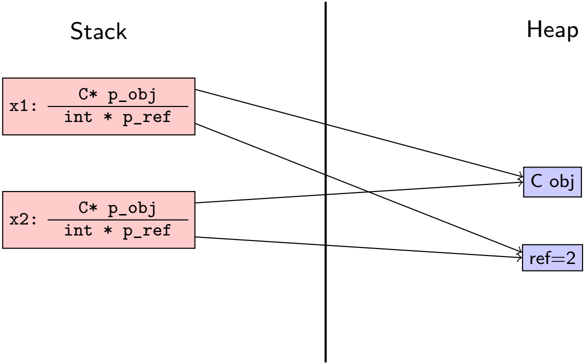


```
std::shared_ptr<C> x1= std::make_shared<C>();
```

Smart pointer schematic

(this is one possibe way to implement it)

```
class C;
```

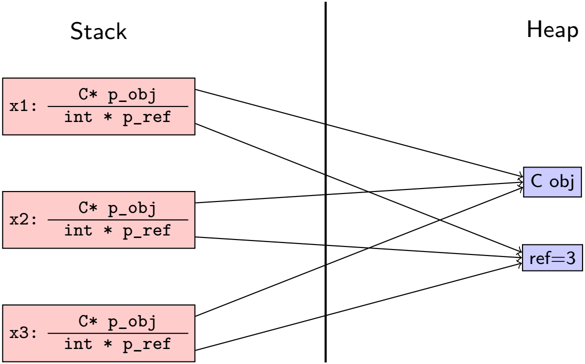


```
std::shared_ptr<C> x1= std::make_shared<C>();
std::shared_ptr<C> x2= x1;
```

Smart pointer schematic

(this is one possibe way to implement it)

```
class C;
```



```
std::shared_ptr<C> x1= std::make_shared<C>();
std::shared_ptr<C> x2= x1;
std::shared_ptr<C> x3= x1;
```

C++ code using vectors, C-Style, with data on stack

File /net/wir/examples/part1/c-style-stack.cxx

```
#include <cstdio>

void initialize(double *x, int n)
{
    for (int i=0;i<n;i++) x[i]= 1.0/(double)(1+n-i);
}

double sum_elements(double *x, int n)
{
    double sum=0;
    for (int i=0;i<n;i++) sum+=x[i];
    return sum;
}

int main()
{
    const int n=1.0e7;
    double x[n];
    initialize(x,n);
    double s=sum_elements(x,n);
    printf("sum=%e\n",s);
}
```

- Large arrays may not fit on stack
- C-Style arrays do not know their length

C++ code using vectors, C-Style, with data on heap

File /net/wir/examples/part1/c-style-heap.cxx

```cpp
#include <cstdio>
#include <cstdlib>
#include <new>

// initialize vector x with some data
void initialize(double *x, int n)
{
    for (int i=0;i<n;i++) x[i]= 1.0/(double)(1+n-i);
}

// calculate the sum of the elements of x
double sum_elements(double *x, int n)
{
    double sum=0;
    for (int i=0;i<n;i++) sum+=x[i];
    return sum;
}

int main()
{
    const int n=1.0e7;
    try  {  x=new double[n]; // allocate memory for vector on heap }
    catch (std::bad_alloc)  { printf("error allocating x\n");  exit(EXIT_FAILURE); }
    initialize(x,n);
    double s=sum_elements(x,n);
    printf("sum=%e\n",s);
    delete[] x;
}
```

- C-Style arrays do not know their length
- Proper memory management is error prone

C++ code using vectors, (mostly) modern C++-style

File `/net/wir/examples/part1/cxx-style-ref.cxx`

```cpp
#include <cstdio>
#include <vector>

void initialize(std::vector<double>& x)
{
    for (int i=0;i<x.size();i++) x[i]= 1.0/(double)(1+n-i);
}

double sum_elements(std::vector<double>& x)
{
    double sum=0;
    for (int i=0;i<x.size();i++)sum+=x[i];
    return sum;
}

int main()
{
    const int n=1.0e7;
    std::vector<double> x(n); // Construct vector with n elements
                              // Object "lives" on stack, data on heap
    initialize(x);
    double s=sum_elements(x);
    printf("sum=%e\n",s);
    // Object destructor automatically called at end of lifetime
    // So data array is freed automatically
}
```

- Heap memory management controlled by object lifetime
- Recommended style *if we can completely stay within C++*

C++ code using vectors, (mostly) modern C++-style with smart pointers

File /net/wir/examples/part1/cxx-style-sharedptr.cxx

```cpp
#include <cstdio>
#include <vector>
#include <memory>

void initialize(std::vector<double> &x)
{
    for (int i=0;i<x.size();i++) x[i]= 1.0/(double)(1+n-i);
}

double sum_elements(std::vector<double> & x)
{
    double sum=0;
    for (int i=0;i<x.size();i++)sum+=x[i];
    return sum;
}

int main()
{
    const int n=1.0e7;
    // call constructor and wrap pointer into smart pointer
    auto x=std::make_shared<std::vector<double>>(n);
    initialize(*x);
    double s=sum_elements(*x);
    printf("sum=%e\n",s);
    // smartpointer calls desctrutor if reference count
    // reaches zero
}
```

- Heap memory management controlled by smart pointer lifetime
- If method or function does not store the object, pass by reference ⇒ API stays the same as for previous case.

Floating point representation

- Scientific notation of floating point numbers: e.g. $x = 6.022 \cdot 10^{23}$
- Representation formula:

$$x = \pm \sum_{i=0}^{\infty} d_i \beta^{-i} \beta^e$$

  - $\beta \in \mathbb{N}, \beta \geq 2$: base
  - $d_i \in \mathbb{N}, 0 \leq d_i \leq \beta$: mantissa digits
  - $e \in \mathbb{Z}$: exponent

- Representation on computer:

$$x = \pm \sum_{i=0}^{t-1} d_i \beta^{-i} \beta^e$$

  - $\beta = 2$
  - t: mantissa length, e.g. $t = 53$ for IEEE double
  - $L \leq e \leq U$, e.g. $-1022 \leq e \leq 1023$ (10 bits) for IEEE double
  - $d_0 \neq 0 \Rightarrow$ normalized numbers, unique representation

Floating point limits

- symmetry wrt. 0 because of sign bit
- smallest positive normalized number: $d_0 = 1, d_i = 0, i = 1 \ldots t-1$
  $x_{min} = \beta^L$
- smallest positive denormalized number: $d_i = 0, i = 0 \ldots t-2, d_{t-1} = 1$
  $x_{min} = \beta^{1-t}\beta^L$
- largest positive normalized number: $d_i = \beta - 1, 0 \ldots t-1$
  $x_{max} = \beta(1 - \beta^{1-t})\beta^U$

Machine precision

- Exact value x
- Approximation $\tilde{x}$
- Then: $|\frac{\tilde{x}-x}{x}| < \epsilon$ is the best accuracy estimate we can get, where
  - $\epsilon = \beta^{1-t}$ (truncation)
  - $\epsilon = \frac{1}{2}\beta^{1-t}$ (rounding)
- Also: ϵ is the smallest representable number such that $1 + \epsilon > 1$.
- Relative errors show up in partiular when
  - subtracting two close numbers
  - adding smaller numbers to larger ones

Starting on unix

Some shell commands in the terminal window

```
| ls -l                       | list files in directory                               |
|                             | subdirectories are marked with ’d’                    |
|                             | in the first column of permission list                |
| cd  dir                     | change directory to dir                               |
| cd  ..                      | change directory one level up in directory hierachy   |
| cp  file1 file2             | copy file1 to file2                                   |
| cp  file1 dir               | copy file1 to directory                               |
| mv  file1 file2             | rename file1 to file2                                 |
| mv  file1 dir               | move file1 to directory                               |
| rm  file                    | delete file                                           |
| [cmd] *.o                   | perform command on all files with name ending with .o |
```

Editors & IDEs

- Source code is written with text editors
  (as compared to word processors like `MS Word` or `libreoffice`)
- Editors installed are
  - `gedit` - text editor of gnome desktop (recommended)
  - `emacs` - comprensive, powerful, a bit unusual GUI (my preferred choice)
  - `nedit` - quick and simple
  - `vi`, `vim` - the UNIX purist's crowbar
    (which I avoid as much as possible)
- Integrated development environments (IDE)
  - Integrated editor/debugger/compiler
  - eclipse (need to get myself used to it before teaching)

Command line instructions to control compiler

- By default, the compiler command performs the linking process as well
- Compiler command (Linux)

```
| g++     | GNU C++ compiler                 |
| clang++ | CLANG compiler from LLVM project |
| g++-5   | GNU C++ 5.x                      |
| icpc    | Intel compiler                   |
```

- Options (common to all of those named above, but not standardized)

```
| -o name              | Name of output file                |
| -g                   | Generate debugging instructions    |
| -O0, -O1, -O2, -O3   | Optimization levels                |
| -c                   | Avoid linking                      |
| -I<path>             | Add <path> to include search path |
| -D<symbol>           | Define preprocessor symbol         |
| -std=c++11           | Use C++11 standard                 |
```

Obtaining and compiling the examples

- Copy files, creating subdirectory part1
  - the . denotes the current directory

```
$ cp -r /net/wir/examples/part1 .
```

- Compile sources (for each of the .cxx files)

```
$ g++ --std=c++11 -o executable source.cxx
```

- Run executable

```
$ ./executable
```

How to copy stuff to your computer

- On Mac, Linux, use ssh:

```
$ scp -r wir-1xy@unixpool.math.tu-berlin.de:/wir/net/examples/part1 .
```

- On Windows
  - install cygwin
  - us WinSCP