C++ tricks

Useful includes

```
#include <stdlib.h>
#include <string>   // for string, substring
#include <sstream>  // for stringstream
#include <iostream> // for cout, puts etc.
#include <stdio.h>  // for  sprintf, printf( "%lf\n", accum );
#include <time.h>   // time
#include <fcntl.h>  // open()
#include <unistd.h> // read()
#include <iomanip>  // for setw
#include <fstream>  // for ifstream
#include <cstring>  // for memcpy, placed in the std namespace
#include <cstddef>  // for size_t
#include <stdint.h>  // uint8_t, uint32_t, etc.

using namespace std; // allows get rid of std:: prefix.
```

Formatting of the string

Formatting of the string

```
double val = 12345.26374859560;
std::stringstream ss; ss << std::setw(8) << std::setprecision(8) << std::right << val;
std::string s = ss.str();
```

Input from stringstream

```
std::string s("121 151 225 456");
std::stringstream ss(s);
T val;
do { ss >> val; cout << val << endl; } while( ss.good() );
```

String methods

```
#include <string>

std::string basename(const std::string& path)
{
  size_t pos = path.find_last_of('/');
  //if (pos == std::string::npos) return std::string();
  if (pos < path.size()-1) return std::string(path,pos+1);
  else return std::string();
}
```

Input from file value by value

```
#include <fstream>  // for ifstream

  std::ifstream f("file.txt");
  std::string val;
  while (f>>val) {
      std::cout << "val: " << val << '\n';
  }
  f.close();
```

Input from file line by line

```
#include <fstream>  // for ifstream

  std::ifstream f("file.txt");
  std::string line;
  while (std::getline(f, line)) {
      std::cout << "line: " << line << '\n';
  }
  f.close();
```

Print long space or repeated character

> *std::cout << std::setw(80) << " " << std::setw(4);*

> *std::cout << std::setw(20) << setfill('=') << '=';*

Bit operations

Check if the n-th bit is on: `if( m_print_bits & (1 << n) ) ...`

Set n-th bit: `mask |=  (1 << n)`

Clear n-th bit: `bit_fld &= ~(1 << n)`

Toggle (swap 0<->1) n-th bit: `bit_fld ^= (1 << n)`

Check if the element is in the vector or list

```
  T element = <instatiation>;
  std::vector<T> v; v=<initialization>;
  if (std::find(v.begin(), v.end(), element) != v.end()) {/*element is in v*/}
  else {/* is not there */}
```

Add the element to the vector or list if it is not there

```
  T element = <instatiation>;
  std::vector<T> v; v=<initialization>;
  if (std::find(v.begin(), v.end(), element) == v.end()) v.push_back(element);
```

Time

```
#include <stdio.h> // for printf
#include <time.h>
#include <iostream> // for cout
using namespace std;

int main ()
{
  // Get current time in seconds since January 1, 1970:
  time_t seconds;
  seconds = time (NULL);
  cout << seconds << " sec  since January 1, 1970\n";

  // Another form of the time access method:
  time_t  time_sec;
  time ( &time_sec );
  struct tm* timeinfo; timeinfo = localtime ( &time_sec );

  cout << "time_sec = " << time_sec << endl;
  cout << "ctime - converts time_sec in char*    : " << ctime (&time_sec);

  char* p_buf = asctime(timeinfo);
  cout << "asctime - converts struct tm* in char*: " << p_buf;

  char c_time_buf[80];  strftime (c_time_buf,80,"%Y-%m-%d %H:%M",timeinfo);
  cout << "strftime - user-formatted time char*  : " << c_time_buf << endl;

  printf ( "struct tm* members:\n");
  printf ( "sec  = %d\n", timeinfo->tm_sec  );
  printf ( "min  = %d\n", timeinfo->tm_min  );
  printf ( "hour = %d\n", timeinfo->tm_hour );
  printf ( "mday = %d\n", timeinfo->tm_mday );
  printf ( "mon  = %d\n", timeinfo->tm_mon  );
  printf ( "year = %d\n", timeinfo->tm_year );
  printf ( "wday = %d\n", timeinfo->tm_wday );
  printf ( "yday = %d\n", timeinfo->tm_yday );
  printf ( "isdst= %d\n", timeinfo->tm_isdst);

  return ( EXIT_SUCCESS );
}
```

Precise time

Use another time structure, which accounts for sec and nsec.

```
#include <time.h>   // time
  struct timespec start, stop;
  int gettimeStatus = clock_gettime( CLOCK_REALTIME, &start );

  printf ( "gettimeStatus: %d \n", gettimeStatus );
  printf ( "start.tv_sec = %d  start.tv_nsec = %d \n", start.tv_sec, start.tv_nsec );

      gettimeStatus = clock_gettime( CLOCK_REALTIME, &stop );
  printf ( " stop.tv_sec = %d   stop.tv_nsec = %d \n", stop.tv_sec,  stop.tv_nsec );
  printf ( "      dt_sec = %d        dt_nsec = %d \n", stop.tv_sec - start.tv_sec,
                                                       stop.tv_nsec- start.tv_nsec);
```

Direct access to command line parameters in C++

```
#include <iostream>
using namespace std;

int main(int argc, char* argv[]) {
   cout << "argc = " << argc << endl;
   for(int i = 0; i < argc; i++)
      cout << "argv[" << i << "] = " << argv[i] << endl;
   return 0;
}
```

Input parameters parsing with getopt

getopt has potential problem: it does not detect the missing argument and pick-up the next available option as an argument...
RETURN VALUE:

- The getopt() function shall return the next option character specified on the command line.
- A colon (':') shall be returned if getopt() detects a missing argument and the first character of optstring was a colon (':').
- A question mark ('?') shall be returned if getopt() encounters an option character not in optstring or detects a missing argument and the first character of optstring was not a colon (':').
- Otherwise, getopt() shall return -1 when all command line options are parsed.

```
#include <getopt.h>
#include <stdio.h>
#include <iostream>
//===================
void usage(char* name) {
  std::cout << "Usage: " << name << " [-a <aaa>] [-b <bbb>] [-c <ccc>] [-h] <p1> [<p2> [<p3> ...]] \n";
}
//===================
int main (int argc, char **argv)
{
    char *avalue = NULL;
    char *bvalue = NULL;
    char *cvalue = NULL;
    int index;
    int c;
    extern char *optarg;
    extern int optind, optopt, opterr;

    while ((c = getopt (argc, argv, ":a:b:c:h")) != -1)
        switch (c)
        {
          case 'a':
            avalue = optarg;
            printf ("a: avalue = %s\n",avalue);
            break;
          case 'b':
            bvalue = optarg;
            printf ("b: bvalue = %s\n",bvalue);
            break;
          case 'c':
            cvalue = optarg;
            printf ("c: cvalue = %s\n",cvalue);
            break;
          case 'h':
            printf ("h: ");
            usage(argv[0]);
            break;
          case ':':
            printf ("(:) Option -%c requires an argument.\n", optopt);
            usage(argv[0]);
            return EXIT_FAILURE;
          case '?':
            printf ("?: Option -%c requires an argument.\n", optopt);
            usage(argv[0]);
            return EXIT_FAILURE;
          default:
            printf ("default: You should not get here... option -%c .\n", optopt);
            abort ();
        }

    printf ("End of options: avalue = %s, bvalue = %s, cvalue = %s\n",
            avalue, bvalue, cvalue);

    for (index = optind; index < argc; index++)
        printf ("Non-option argument %s\n", argv[index]);
    return EXIT_SUCCESS;
}
```

Check if directory exists

```
#include <sys/stat.h>

  bool dir_exists(const std::string& dir)
  {
    struct stat st;
    if( (stat(dir.c_str(),&st) == 0) && (((st.st_mode) & S_IFMT) == S_IFDIR) ) return true;
    return false;
  }
```

Check if file exists

```
#include <fstream>  // for ifstream

  std::ifstream f("file.txt");
  if(not f.good()) ...
```

Iterate over files in the directory

Using `opendir`, `readdir`, `closedir` from STL

```
#include <iostream>
#include <dirent.h>
using namespace std;

int main() {
        DIR*     dir;
        dirent*  pdir;

        dir = opendir(".");      // open current directory
        while (pdir = readdir(dir)) {
                cout << pdir->d_name << endl;
        }
        closedir(dir);
        return 0;
}
```

Using `boost`

```
#include <boost/filesystem.hpp>
namespace fs = boost::filesystem;

  fs::path path = "/reg/d/psdm/CXI/cxi49012/xtc/e158-r0150-s00-c01.xtc";
  fs::path dir  = path.parent_path(); // i.e.: /reg/d/psdm/CXI/cxi49012/xtc

  typedef fs::directory_iterator dir_iter;
  for (dir_iter dit = dir_iter(dir); dit != dir_iter(); ++ dit) {
    std::cout << dit->path().stem().string() << "\n";
  }
```

Other methods of `fs::path`

```
  fs::path path = m_path;
  std::cout << "stem          = " << path.stem().string() << "\n";           // e158-r0150-s00-c00
  std::cout << "extension()   = " << path.extension().string() << "\n";      // .xtc
  std::cout << "filename()    = " << path.filename().string() << "\n";       // e158-r0150-s00-c00.xtc
  std::cout << "parent_path() = " << path.parent_path().string() << "\n";    // /reg/d/psdm/CXI/cxi49012/xtc
```

ndarray and shared pointer

Example of how to access ndarray data in psana module through the shared pointer and construct `ndarray<const T,NDim>`

```
        shared_ptr< ndarray<T,NDim> > shp = evt.get(m_source, m_key, &m_src);
        if (shp.get()) {

          ndarray<T,NDim>* pnda = shp.get();
          ndarray<const T,NDim> nda_const(pnda->data(), pnda->shape());

          ndarray<const T,NDim> nda_const(shp->data(), shp->shape());
        }
```

Singleton

InputParameters.h :

```
#include <iostream> // for cout

class InputParameters  {
public:
  static InputParameters* instance();
  void print();

private:
  InputParameters () ;                // !!!!! Private so that it can not be called from outside
  virtual ~InputParameters () {};

  static InputParameters* m_pInstance; // !!!!! Singleton instance

  // Copy constructor and assignment are disabled by default
  InputParameters ( const InputParameters& ) ;
  InputParameters& operator = ( const InputParameters& ) ;
};
```

InputParameters.cc :

```
#include "<Package>/InputParameters.h"

InputParameters* InputParameters::m_pInstance = NULL; // !!!!!! make global pointer !!!!!

//----------------
InputParameters::InputParameters ()
{
  std::cout << "!!!!!!!! Single instance for singleton class InputParameters is created \n";
}
//----------------
InputParameters* InputParameters::instance()
{
  if( !m_pInstance ) m_pInstance = new InputParameters();
  return m_pInstance;
}
//----------------
void InputParameters::print() {  std::cout << "InputParameters::print()\n"; }
```

Calls:

```
#include "<Package>/InputParameters.h"
int main(int argc, char *argv[])
{
  InputParameters::instance()->print("xyz...");
}
```

Shortcuts:

```
#define PRINT InputParameters::instance()->print
int main(){
  PRINT("This is a line of log");
  return 0;
}
```

Declaration, Initialization, Definition of static const parameters

```
// *.h :
class A {
public:

  static const int     INT_PAR    = 185;
  static const double  DOUBLE_PAR;
}

// *.cpp :
const int    A::INT_PAR;
const double A::DOUBLE_PAR = 109.92;
```

Issue OS command from C++

Use method system

```
// int system (const char* command);

#include <stdlib.h>     /* system, NULL, EXIT_FAILURE */
int status = system ("ls -l");
```

Use method popen

```
#include <string>
#include <iostream>
#include <stdio.h>

std::string exec(char* cmd) {
    FILE* pipe = popen(cmd, "r");
    if (!pipe) return "ERROR";
    char buffer[128];
    std::string result = "";
    while(!feof(pipe)) {
            if(fgets(buffer, 128, pipe) != NULL)
                    result += buffer;
    }
    pclose(pipe);
    return result;
}
```

Valgrind - analysis of job memory consumption

```
valgrind --tool=massif --time-unit=B psana -n 1 -m ImgAlgos.PixCoordsProducer exp=cxi86415:run=62
```

this command runs the job and produces the file like `massif.out.23864`, which can be printed by command:

```
ms_print massif.out.23864
```

Building C/C++ library using distutils.core setup, Extension

http://stackoverflow.com/questions/16854066/using-distutils-and-build-clib-to-build-c-library

Instead of passing a library name as a string, pass a tuple with the sources to compile:

```
# setup.py
#__________

import sys
from distutils.core import setup
from distutils.command.build_clib import build_clib
from distutils.extension import Extension
from Cython.Distutils import build_ext

libhello = ('hello', {'sources': ['hello.c']})  # <<<<==============

ext_modules=[
    Extension("demo", ["demo.pyx"])
]

setup(
        name = 'demo',
        libraries = [libhello],
        cmdclass = {'build_clib': build_clib, 'build_ext': build_ext},
        ext_modules = ext_modules
)
#__________
```

Using C++ in Cython