# Tillämpad programmering


C++ make, configure och arrayer

Johan Montelius

# C++ sndcopy.cc


```
#include <stdio>

#include “sndreader.h”
#incldue “sndwrite.h”

int main(int argc, char *argv[]) {

        :
        :
```

# C++ sndreader.h


```
#include <iostream>

// Import a C header file
extern "C" {
#include <sndfile.h>
}

#ifndef __SNDREADER_H__
#define __SNDREADER_H__
    :
    :
```

# C++ sndfile.h

```
enum {      /* Major formats. */
      SF_FORMAT_WAV                     = 0x010000,
      /* Microsoft WAV format (little endian default). */
      SF_FORMAT_AIFF                    = 0x020000,
      /* Apple/SGI AIFF format (big endian). */
      SF_FORMAT_AU                      = 0x030000,
      /* Sun/NeXT AU format (big endian). */
      SF_FORMAT_RAW                     = 0x040000,
      /* RAW PCM data. */
      :
      :
```

# C++ sndfile.h

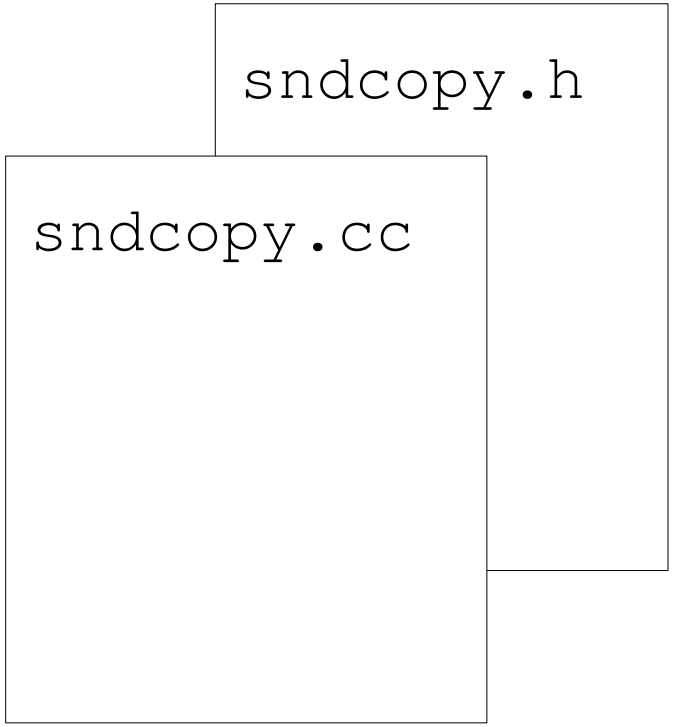
```
struct SF_INFO {
      sf_count_t  frames ; /* Used to be called ...samples. */
      int                 samplerate ;
      int                 channels ;
      int                 format ;
      int                 sections ;
      int                 seekable ;
};

typedef struct SF_INFO SF_INFO ;

      :
```

# C++ sndfile.h

```
SNDFILE* sf_open (const char *path,int mode,SF_INFO *sfinfo);
```

# g++


KTH
VETENSKAP
OCH KONST


sndcopy.h
sndcopy.cc


sndfile.h


libsndfile.so

# kompilera och länka


```
g++
    -o ut-fil
    -c enbart kompilering
    -E enbart förbehandling
    -I sökväg för header
    -L sökväg för bibliotek
    -l bibliotek
    källkodsfil
```

# libsndfile-1.0.25.tar


```
>tar -xvf ibsndfile-1.0.25.tar
libsndfile-1.0.25/
libsndfile-1.0.25/NEWS
libsndfile-1.0.25/M4/
        :
        :
```

# libsndfile-1.0.25

```
>ls
AUTHORS
    :
COPYING
INSTALL
    :
README
    :
```

# README

```
LINUX
-----
Whereever possible, you should use
the packages supplied by your Linux
distribution.

If you really do need to compile
from source it should be as easy as:

            ./configure
            make
            make install
```

# INSTALL

```
The simplest way to compile this package is:

  1. `cd' to the directory containing
     the package's source code and type
     `./configure'
  2. Type `make' to compile the package.
  3. Optionally, type `make check' to run any
     self-tests that come with the package.
  4. Type `make install' to install the programs
     and any data files and documentation.
```

# INSTALL

```
  By default, `make install' will install
the package's files in

     `/usr/local/bin',
     `/usr/local/man',

etc. You can specify an installation prefix other
than `/usr/local' by giving `configure' the
option

     `--prefix=PATH'.
```

# /configure

```
>./configure --prefix=/home/user/tmp
checking build system type... x86_64-unknown-linux-gnu
checking host system type... x86_64-unknown-linux-gnu
checking target system type... x86_64-unknown-linux-gnu
checking for a BSD-compatible install... /usr/bin/instal
checking whether build environment is sane... yes
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
      :
      :
```

# libsndfile-1.0.25

```
>ls
    :
 Makefile
    :
```

# make

```
>make
      :
      :
make[2]: Entering directory ../src
  CC     sndfile.lo
  CC     aiff.lo
  CC     au.lo
      :
      :
```

# make install

```
>make install
      :
      :
Libraries have been installed in:
   /home/user/tmp/lib
      :
      :
```

# /home/user/tmp


```
/home/user/tmp/
        bin
        include/
                sndfile.h
        lib/
                libsndfile.so
        libsndfile-1.0.25
        share
```

# 2-1.zip


```
>wget https:// ..... 2-1.zip
  :
  :
>unzip 2-1.zip
  :
  :
>ls 2-1
      Makefile
      sndcopy.cc
      sndreader.h
      sndwriter.h
      wavfiles
```

# Makefile

```
CXX         = g++
CXXFLAGS    = -g
LDFLAGS     = -L/usr/local/lib  \
              -lsndfile
```

# Makefile

```
CXX        = g++
CXXFLAGS   = -g -I/home/user/tmp/include
LDFLAGS    = -L/usr/local/lib  \
             -L/home/user/tmp/lib  \
             -lsndfile
```

# Makefile

```
#
# Programs to be created...
#
PROGSRC    = sndcopy.cc
PROGEXE    = $(PROGSRC:%.cc=%.exe)

all: $(PROGEXE)
```

# Makefile

```
%.exe: %.cc sndreader.h sndwriter.h
	$(CXX) $(CXXFLAGS) $< -o $@ $(LDFLAGS)
```

```
g++  -g -I/home/user/tmp/include \
     sndcopy.cc  \
     -o sndcopy.exe \
     -L/usr/local/lib \
     -L/home/user/tmp/lib
```

# att bygga


- configure
  - söker efter rätta verktyg
  - skapar en Makefile som är skräddarsydd
- make
  - håller rätt på beroende
  - kompilerar endast det som är nödvändigt
- kompilering
  - förbehanding (preprocessor)
  - kompilering
  - länkning

# primitiva arrayer (C)


```
int a[4];

int b[] = {1,2,3,4};

double c[100];

long d[10][10];
```

# char a[]


```
char a[5];

char b[] = {'H', 'e', 'l', 'l', 'o'};

char c[] = “Hello”;

cout << a << endl;
cout << b << endl;
cout << c << endl;
```

# som argument


```
void foo(int (& a)[10]) {
     a är en array
}

void bar(int (*a)[10]) {
     a är en pekare till en array
}
```

# inte så bra


```
void zot(int a[]) {
     a är en pekare till första
     elementet i en array av okänd
     storlek
}

void grk(int *a) {
     a är en int pekare
}

void baz(int a[10]) {
     a är en pekare till första
     elementet i en array av storlek 10
}
```

# new int[n]


```
int n = 10;
      :
int *a = new int a[n];
      :
zot(a, n);
      :
```

# array<Type, Size>


```
typedef array<int,10> MyArray;

int main() {
      MyArray *arr = new MyArray;
      foo(arr);
}
```

# array<Type, Size>

```
void foo(MyArray *a) {

     int size = (a*).size()

     MyArray::iterator begin = (a*).begin();
     MyArray::iterator end = (a*).end();
     MyArray::iterator p;

     for(p = begin, p != end, p++) {
          cout << *p << endl;
     }
}
```

# vector<Type>

```
typedef vector<int> MyVector;

int main() {

     MyVector *vec = new MyVector;

     (*vec).push_back(10);
     (*vec).push_back(11);
     (*vec).push_back(12);
     (*vec).push_back(13);

     foo(vec);
}
```

# array<Type, Size>

```
void foo(MyVector *v) {

     int size = (v*).size()

     for(int i = 0; i < size; i++) {
          cout << (*v)[i] << endl;
     }
}
```

# tumregler

- förstå c arrayer
- använd vector