An overview of Unitex C/C++ code

Sébastien Paumier

LIGM, Université Paris-Est

Golden rules

- must be portable:
  - must not depend on a specific compiler
  - must not depend on a specific processor (endianness, 32/64 bits)
  - must not depend on a specific system
- must be thread-safe
- preserve compatibility with previous versions of programs

General tools

The core: Unicode.cpp/.h

- basic type: `unichar`
- file type: `U_FILE`
  - encapsulates encoding information
  - allows virtual file support
- string functions: `u_strcpy`, `u_strcat`, ...
- I/O functions: `u_printf`, `u_fopen`, ...

FileEncoding.h

- every Unitex program can change the input/output encodings with `-k`/`-q`
  - `VersatileEncodingConfig`
- file opening:

```
VersatileEncodingConfig vec=VEC_DEFAULT;
...
U_FILE* res=u_fopen(vec,name,U_WRITE);
```

getopt

- for thread-safety, use `UnitexGetOpt.h`

```
int val,index=-1;
struct OptVars* vars=new_OptVars();
while (EOF!=(val=getopt_long_TS(argc,argv,optstring_Compress,lopts_Compress,&index,vars))) {
   switch(val) {
     ...
   case 'h': usage(); return 0;
   case ':': if (index==-1) fatal_error("Missing argument for option -%c\n",vars->optopt);
             else fatal_error("Missing argument for option --
%s\n",lopts_Compress[index].name);
   case 'k': if (vars->optarg[0]=='\0') {
                fatal_error("Empty input_encoding argument\n");
             }
             decode_reading_encoding_parameter(&(vec.mask_encoding_compatibility_input),vars-
>optarg);
             break;
   case 'q': if (vars->optarg[0]=='\0') {
                fatal_error("Empty output_encoding argument\n");
             }
             decode_writing_encoding_parameter(&(vec.encoding_output),&(vec.bom_output),vars-
>optarg);
             break;
   case '?': if (index==-1) fatal_error("Invalid option -%c\n",vars->optopt);
             else fatal_error("Invalid option --%s\n",vars->optarg);
             break;
   }
   index=-1;
}
```

Ustring.h

- `Ustring` is a self-resizable string
- useful to avoid buffer overflow
- prefer it to `unichar*`
- many functions use it:
  - `u_strcpy`, `u_strcat`, ...
  - `int readline(Ustring*,U_FILE*);`
  - `unichar* readline_safe(U_FILE* f);`

Lists

- `List_int.h`, `List_pointer.h`, `List_ustring.h`

```
struct list_int* new_list_int(int,struct list_int*,Abstract_allocator
prv_alloc=STANDARD_ALLOCATOR);
struct list_int* new_list_int(int,Abstract_allocator
prv_alloc=STANDARD_ALLOCATOR);
void free_list_int(struct list_int*);
void free_list_int(struct list_int*,Abstract_allocator prv_alloc);
struct list_int* sorted_insert(int,struct list_int*,Abstract_allocator
prv_alloc=STANDARD_ALLOCATOR);
int is_in_list(int,const struct list_int*);
int equal_list_int(const struct list_int*,const struct list_int*);
...
```

invoked without the last parameter=normal malloc/free, but you can use your own allocator if you want

Hash tables

- `HashTable.h`: open hashing supporting multiple types defined in `Any.h`
- `get_value`:
  - gets a `struct any` containing the value associated to an element or `NULL` if not found
  - you can associate `NULL` to a key
  - you can decide whether to insert a value or not when the key is not found

Vectors, stacks and FIFOs

- auto-resizable arrays defined in `Vector.h`:
  - `vector_ptr`, `vector_int`, `vector_float`, `vector_double`
  - `new_vector_XXX`, `free_vector_XXX`
  - `vector_XXX_add`, `vector_XXX_copy`, `vector_XXX_add_if_absent`
  - `vector_XXX_contains`
  - …
- `Stack_int.h`, `Stack_pointer.h`, `Stack_unichar.h`, `FIFO.h`

String_hash.h

- `struct string_hash`=structure used to map *N* strings to integers in *[0;N-1]*
- uses a lexicographic tree
- string → index:
  - `get_value_index` functions
- index → string:
  - use the field `unichar** value;`

SingleGraph.h

- type `SingleGraph`: finite-state automaton with transitions that can be tagged either by integers or pointers
- used by `Grf2Fst2`, `Elag`, ...
- supports many standard operations:
  - determinization, minimization, pruning, topological sort, ...
- do not use Elag specific files: `AutDeterminization.h`, `AutMinimization.h`, etc

Bit manipulation

- `BitArray.h`:
  - bit arrays to store 1-bit, 2-bits or 4-bits information in a compact way
- `BitMasks.h`:
  - just to make sure that basic bit operations are done once and well to avoid painful bug tracking

Buffer.h

- `struct buffer`: buffering `unichars` or `ints` from a file
  - for text: loads `\r\n` as a single `\n`
- prefer file mapping if possible:
  - `ABSTRACTMAPFILE` in `Af_stdio.h`

StringParsing.h

- `parse_string` functions:
  - looking for a stop symbol set
  - can keep some chars protected
  - can forbid some chars to appear
- `escape`: protect characters with a \
- `unprotect`: string copy that unprotects chars that belong to a given set

Errors

- always use functions from `Error.h`
- fail policy: any inconsistent internal state must lead to a `fatal_error`
- all memory allocation errors must lead to a `fatal_alloc_error`
- never call `exit` yourself!!

File.h

- basic file name operations:
  - `get_path`, `remove_extension`, `is_absolute_path`, `is_root`, ...
- some file operations:
  - `copy_file`, `fexists`, `create_path_to_file`, ...

Specific data structures

Alphabet.h

- `Alphabet*`: defines the characters to be considered as letters
- defines the allowed case variations (ex: in French `E` match `é` and `e`, and `É` matches `é` as well):
  - `is_upper_of`, `is_equal_or_uppercase`, `is_letter`, ...
- a `NULL Alphabet*` means that the Unicode letter definition will be used

DELA.h

- `struct dela_entry`: DELAF dictionary entries like:

```
eaten,eat.V:K
```

- many functions to build and manipulate such data:
  - `tokenize_DELAF_line`, `check_tag_token`, `same_codes`, `dic_entry_contain_gram_code`, ...

Fst2.h

- `Fst2*`: a compiled grammar described as a set of graphs that can call each other, with common transition tags
- all states are stored in an array
- `int* initial_states` indicates where each graph begins

Grf_lib.h

- the `Grf` structure can be used to manipulate a .grf file with all its graphical information
- used by `GrfBeauty.h` and `GrfSvn_lib.h` used by the `GrfDiff`/`GrfDiff3` tools

Tfst.h

- `Tfst`: a text automaton, with only one sentence loaded at once
- open you have obtained it with `open_text_automaton`, you can browse the text automaton with `load_sentence`

How to create a Unitex program

Main_XXX.cpp

- Unitex programs must be compiled together into `UnitexToolLogger`, so we need to isolate `main` functions into `Main_XXX.cpp` files:

```
#include "IOBuffer.h"
#include "XXX.h"

using namespace unitex;

int main(int argc,char* argv[]) {
/* Every Unitex program must start by this instruction,
 * in order to avoid display problems when called from
 * the graphical interface */
setBufferMode();

return main_XXX(argc,argv);
}
```

Main_XXX.h

- `xxx.h` declares the real `main` as well as getopt things:

```
#ifndef XXXH
#define XXXH

#include "UnitexGetOpt.h"

#ifndef HAS_UNITEX_NAMESPACE
#define HAS_UNITEX_NAMESPACE 1
#endif

namespace unitex {

extern const char* optstring_XXX;
extern const struct option_TS lopts_XXX[];
extern const char* usage_XXX;

int main_XXX(int argc,char* const argv[]);

} // namespace unitex

#endif
```

XXX.cpp

```
#ifndef HAS_UNITEX_NAMESPACE
#define HAS_UNITEX_NAMESPACE 1
#endif

namespace unitex {

const char* usage_XXX =
         "Usage: XXX [OPTIONS] ...";

static void usage() {
u_printf("%S",COPYRIGHT);
u_printf(usage_XXX);
}

const char* optstring_XXX=":hk:q:";
const struct option_TS lopts_XXX[]= {
      {"help",no_argument_TS,NULL,'h'},
      {"input_encoding",required_argument_TS,NULL,'k'},
      {"output_encoding",required_argument_TS,NULL,'q'},
      {NULL,no_argument_TS,NULL,0}
};
```

XXX.cpp

```
int main_XXX(int argc,char* const argv[]) {
if (argc==1) {
   usage();
   return 0;
}
VersatileEncodingConfig vec=VEC_DEFAULT;
int val,index=-1;
struct OptVars* vars=new_OptVars();
while (EOF!
=(val=getopt_long_TS(argc,argv,optstring_XXX,lopts_XXX,&index,vars))) {
   switch(val) {
   case 'h': usage(); return 0;
   case 'k': ...
   case 'q': ...
   case ':': ...
   case '?': ...
   }
   index=-1;
}
...
free_OptVars(vars);
return 0;
}
```

UnitexTool.cpp

```
#if (((!defined(NO_TOOL_XXX))) || defined(TOOL_XXX))
#include "XXX.h"
#endif
...
#if (((!defined(NO_TOOL_XXX))) || defined(TOOL_XXX))
    { "XXX", 3, &main_XXX, usage_XXX, optstring_XXX, lopts_XXX } ,
#endif
```

strlen of XXX

Makefile

```
XXX       = XXX
XXX_OBJS = Main_XXX.o XXX.o IOBuffer.o Af_stdio.o ActivityLogger.o\
           Unicode.o AbstractAllocator.o Error.o UnitexGetOpt.o
           Persistence.o $(ADDITIONAL_OBJECT) $(SYSLIBMAPPED)\
           $(SYSLIBSYNCTOOL) your own .o ...

UNITEXTOOL_OBJS = Main_UnitexTool.o \
                  ...
                  XXX.o your own .o ...

UNITEXTOOL_LOGGER_NO_MAIN_OBJS = Main_UnitexTool.o \
                  ...
                  XXX.o your own .o ...

PROGS = ... $(XXX) ...

OBJS = ... $(XXX_OBJS) ...

$(BIN_DIR)$(XXX)$(EXTENSION): $(XXX_OBJS)
    $(CC) -o $@ $+ $(OPTIONS)
```