notes

disable warnings in gcc

```
#pragma GCC system_header
```

works great for generated code.

cgdb, highlight current line

```
:hi SelectedLineNr cterm=reverse
```

printing all the gcc defines

```
gcc -dM -E - < /dev/null
```

foldr map and curry in the C preprocessor language

Might as well do scheme next :).

```
//compile assert macro
#define C_ASSERT(test) \
    switch(0) {\
      case 0:\
      case test:;\
    }
// ... means any number of arguments a1 -> a2 -> a3 ... -> an
// so (...) is (a1 -> a2 -> a3 ... -> an) or a list
// but i will use haskells [a] types to express lists
// where lists are expected explicitly

//count arguments
//COUNT_ARGS :: ... -> Int
#define COUNT_ARGS(...) COUNT_ARGS_(,##__VA_ARGS__,6,5,4,3,2,1,0)
#define COUNT_ARGS_(z,a,b,c,d,e,f,cnt,...) cnt

//call an object
//CALL :: (... -> a) -> (...) -> a
#define CALL(x,y) x y

//HEAD :: [a] -> a
```

```
#define HEAD(args) CALL(HEAD_,args)
#define HEAD_(a,...) a

//TAIL :: [a] -> [a]
#define TAIL(args) (CALL(TAIL_,args))
#define TAIL_(a,...) __VA_ARGS__

//UNPACK :: (...) -> ...
#define UNPACK(args) CALL(UNPACK_,args)
#define UNPACK_(...) __VA_ARGS__

#define ISEMPTY(ls) PASTE(ISEMPTY_,CALL_(COUNT_ARGS,PASTE(ISEMPTY_,CALL
#define CALL_(f,a) f(a)
#define ISEMPTY_0 a,b
#define ISEMPTY_2 1
#define ISEMPTY_1 0

//JOIN :: (...) -> (...) -> (...)
#define JOIN(l1,l2) PASTE(JOIN_,PASTE(ISEMPTY(l1),ISEMPTY(l2)))(l1,l2)
#define JOIN_00(l1,l2) (UNPACK(l1),UNPACK(l2))
#define JOIN_10(l1,l2) l2
#define JOIN_01(l1,l2) l1
#define JOIN_11(l1,l2) ()

//curry a function to it arguments
#define CURRY(f,args) JOIN(f,args)

//evaluate a function object
#define EVAL(f) EVAL_ f
#define EVAL_(f,...) f(__VA_ARGS__)


//PASTE :: a -> b -> c
//this is really in the preprocessor domain, so hard to type
#define PASTE(aa,bb) PASTE_(aa,bb)
#define PASTE_(aa,bb) aa ## bb


//FOLDR :: (a -> b -> b) -> b -> [a] -> b
#define FOLDR(func,accum,lst)       PASTE(FOLDR_,CALL(COUNT_ARGS,lst))(
#define FOLDR_1(func,accum,lst)     EVAL_FR(CURRY(func,(HEAD(lst), accu
#define FOLDR_2(func,accum,lst)     EVAL_FR(CURRY(func,(HEAD(lst),FOLDR
#define FOLDR_3(func,accum,lst)     EVAL_FR(CURRY(func,(HEAD(lst),FOLDR
#define FOLDR_4(func,accum,lst)     EVAL_FR(CURRY(func,(HEAD(lst),FOLDR

//the map needs a different eval then the foldr
//i dont fully grok this part yet
#define EVAL_FR(f) EVAL_FR_ f
#define EVAL_FR_(f,...) f(__VA_ARGS__)

//MAP :: (a -> b) -> [a] -> b
#define MAP(func,lst)  FOLDR((MAP_,func),(),lst)
#define MAP_(func,aa,acc)  JOIN((EVAL_MP(CURRY(func,(aa)))), acc)
#define EVAL_MP(f) EVAL_MP_ f
```

```
#define EVAL_MP_(f,...) f(__VA_ARGS__)

int main(void) {
   C_ASSERT(0 == COUNT_ARGS());
   C_ASSERT(1 == COUNT_ARGS(1));
   C_ASSERT(2 == COUNT_ARGS(1,2));
   C_ASSERT(3 == COUNT_ARGS(1,2,3));

   C_ASSERT(12 == PASTE(1,2));

   C_ASSERT(1 == HEAD((1,2,3)));
   C_ASSERT(2 == HEAD(TAIL((1,2,3))));
   C_ASSERT(3 == HEAD(TAIL(TAIL((1,2,3,4)))))

   C_ASSERT(1 == ISEMPTY(()))
   C_ASSERT(0 == ISEMPTY((1)))
   C_ASSERT(0 == ISEMPTY((1,2)))
   C_ASSERT(0 == ISEMPTY((1,2,3)))
   C_ASSERT(1 == ISEMPTY(JOIN((),())))

   C_ASSERT(1 == HEAD(JOIN((),(1))))
   C_ASSERT(1 == HEAD(JOIN((1),())))
   C_ASSERT(1 == HEAD(JOIN((1,2),())))
   C_ASSERT(1 == HEAD(JOIN((1),(2))))
   C_ASSERT(2 == HEAD(JOIN((2),(1))))
   C_ASSERT(1 == HEAD(JOIN((1,2),(3,4))))
   C_ASSERT(2 == HEAD(TAIL(JOIN((1,2),(3,4)))))
   C_ASSERT(3 == HEAD(TAIL(TAIL(JOIN((1,2),(3,4))))))
   C_ASSERT(4 == HEAD(TAIL(TAIL(TAIL(JOIN((1,2),(3,4)))))))

   C_ASSERT(1 == EVAL(CURRY((HEAD),((1,2)))))
   C_ASSERT(12 == EVAL(CURRY((PASTE),(1,2))))
   C_ASSERT(12 == EVAL(CURRY((PASTE,1),(2))))

   C_ASSERT(123 == FOLDR((PASTE),,(1,2,3)))

   #define ID(x) x
   C_ASSERT(1 == HEAD(MAP((ID),(1,2,3))))
   C_ASSERT(1 == HEAD(MAP((ID),MAP((ID),(1,2,3)))))
   C_ASSERT(1 == HEAD(MAP((ID),MAP((ID),MAP((ID),(1,2,3))))))


   return 0;
}
```

counting args with C macros

```
/**
 * we need a comma at the start for ##__VA_ARGS__ to consume then
 * the arguments are pushed out in such a way that 'cnt' ends up with
 * the right count.
```

```
 */
#define COUNT_ARGS(...) COUNT_ARGS_(,##__VA_ARGS__,6,5,4,3,2,1,0)
#define COUNT_ARGS_(z,a,b,c,d,e,f,cnt,...) cnt

#define C_ASSERT(test) \
    switch(0) {\
      case 0:\
      case test:;\
    }

int main() {
   C_ASSERT(0 ==  COUNT_ARGS());
   C_ASSERT(1 ==  COUNT_ARGS(a));
   C_ASSERT(2 ==  COUNT_ARGS(a,b));
   C_ASSERT(3 ==  COUNT_ARGS(a,b,c));
   C_ASSERT(4 ==  COUNT_ARGS(a,b,c,d));
   C_ASSERT(5 ==  COUNT_ARGS(a,b,c,d,e));
   C_ASSERT(6 ==  COUNT_ARGS(a,b,c,d,e,f));
   return 0;
}
```

compile time assert

```
#define C_ASSERT(test) \
    switch(0) {\
      case 0:\
      case test:;\
    }
```

make repl, sort of

```
#run make '$(expression)' to see what make thinks the expression evalua
$$%:;@$(call true)$(info $(call or,$$$*))
```

make + cabal

```
#run make T=target to trigger the build for target exe
O=dist/build
D=dist/setup-config
A=#

ifneq ($(strip $T),)
A=$O/$T/$T
endif

all:$A
```

```
$O/%:$(shell find . -not -path "*dist/build/*" -iname "*.hs") $D
    cabal build
    touch $@

clean:
    cabal clean

$D:$(wildcard *.cabal)
    cabal install --only-dependencies --enable-executable-profiling --e
    cabal configure
    touch $@
```

.vimrc

```
"remembering cursor position between sessions
:set viminfo='10,\"100,:20,%,n~/.viminfo

function! ResCur()
  if line("'\"") <= line("$")
    normal! g`"
    return 1
  endif
endfunction

augroup resCur
  autocmd!
  autocmd BufWinEnter * call ResCur()
augroup END

:syntax on
:set nowrapscan
:set autoindent
:set expandtab
:set tabstop=3
:set ruler
:set shiftwidth=3
:set ignorecase
:set smartcase
:set tags=./tags;
:set noswapfile
:map <tab> <C-w><C-w>
:set wildmode=longest,list,full
:set wildmenu
"underline spell check
:hi clear SpellBad
:hi SpellBad cterm=underline
"sane highlights for vimdiff
highlight DiffAdd term=reverse cterm=bold ctermbg=green ctermfg=white
highlight DiffChange term=reverse cterm=bold ctermbg=cyan ctermfg=black
highlight DiffText term=reverse cterm=bold ctermbg=gray ctermfg=black
highlight DiffDelete term=reverse cterm=bold ctermbg=red ctermfg=black
```

```
"use tabs in linux kernel code
:au BufNewFile,BufRead /*kernel* set tabstop=8
:au BufNewFile,BufRead /*kernel* set shiftwidth=8
:au BufNewFile,BufRead /*kernel* set noexpandtab
```