```
#include <R.h>
#include <Rinternals.h>
```

C type of all R objects

Quick R API cheat sheet by Simon Urbanek, 3/2012

TYPEOF(SEXP x) LENGTH(SEXP x) isNull(x)

native vectors

type	*payload*	*accessor*	*scalar constructor, coercion to scalar, type check*
INTSXP	int*	INTEGER(x)	ScalarInteger, asInteger, isInteger
REALSXP	double*	REAL(x)	ScalarReal, asReal, isReal [tools: ISNA(x), ISNAN(x), R_FINITE(x)]
LGLSXP	int*	LOGICAL(x)	ScalarLogical, asLogical, isLogical
CPLXSXP	Rcomplex*	COMPLEX(x)	ScalarComplex, asComplex, isComplex
RAWSXP	Rbyte*	RAW(x)	ScalarRaw, --, TYPEOF(x) == RAWSXP

other vectors

VECSXP VECTOR_ELT(x, index) SET_VECTOR_ELT(x, index, value) any
STRSXP STRING_ELT(x, index) SET_STRING_ELT(x, index, value) CHARSXP
SEXP mkString(const char *)
CHARSXP *(payload of STRSXP only!)*
SEXP mkChar(const char *), SEXP mkCharCE(const char*, {CE_NATIVE | CE_UTF8 | CE_LATIN1})
const char *CHAR(x), const char* translateCharUTF8(x)

alloc

allocation/coercion
allocVector(type, length) allocMatrix(type, m, n) mkNamed(type, const char **names {..., ""})
duplicate(x) coerceVector(x, type)

out

stop/warning/output (printf-style arguments)
error(format, ...) warning(format, ...) Rprintf(format, ...) REprintf(format, ...)

globals, sym

most common global variables
R_NilValue = NULL R_GlobalEnv = .GlobalEnv R_NaString = NA_character_
R_NaInt = NA_integer_ R_NaReal = NA_real_ R_NaN, R_PosInf, R_NegInf
symbols (only most common - see Rinternals.h for the full list)
R_NamesSymbol, R_DimSymbol, R_DimNamesSymbol, R_RowNamesSymbol, R_ClassSymbol
any other symbol: install(const char* symbol_name)

attr

attributes
setAttrib(x, symbol, value) getAttrib(x, symbol)

pairlists

pairlists
LISTSXP: [CAR = payload, TAG = symbol/name, CDR = next LISTSXP or R_NilValue (=end)]
constructor: CONS(car=payload, cdr=next)
list1(e1) ~ CONS(e1, R_NilValue) list2(e1, e2) ~ CONS(e1, CONS(e2)) list3 ...
LANGSXP = LISTSXP with language objects, LCONS ~ CONS, lang1 ~ list1, lang2 ~ list2, ...
example: SEXP rnorm = install("rnorm"), x = eval(lang2(rnorm, ScalarInteger(10)), R_GlobalEnv);

protection

protection
GC can be run at any *allocation* so R objects must be protected in some way if they are to be kept across any additional allocations. Protect is a stack so the number of PROTECT calls must match *count* in UNPROTECT. Objects contained in other objects are automatically protected by the enclosing object.
PROTECT(x) UNPROTECT(count) R_PreserveObject(x) R_ReleaseObject(x)

silly example

.Call("replicate_to_list", x, n): *NOTE: Must **always** return a valid value! Even if it is R_NilValue*

```
SEXP replicate_to_list(SEXP x, SEXP N) {
        int n = asInteger(N);
        if (n < 0) error("N must be non-negative");
        SEXP res = allocVector(VECSXP, n);
        for (int i = 0; i < n; i++) SET_VECTOR_ELT(res, i, x);
        return res;
}
```

```
$ R CMD SHLIB foo.c
> dyn.load("foo.so") # or foo.dll
```