

```

/* singlist */
/* lang=C++11 */

/* regular (no-frills) pointer queue (not-circular) */

#ifndef CF_DEBUGS
#define CF_DEBUGS 0 /* compile-time debugging */
#endif

/* revision history:

    = 2013-03-03, David A>D> Morano
    Originally written for Rightcore Network Services.

*/

/* Copyright ' 2013 David A>D> Morano. All rights reserved. */

/*****

    This is a container object (elements are stored within it). This also is
    implemented as a single-linked list of nodes. This object is very useful
    for normal queue operations (insert at tail, remove at head). The
    following operations are supported:

    + instail          insert at tail
    + inshead          insert at head
    + remhead          remove from head

    The only major operation (for relatively normal queues) which is *not*
    supported is:

    + remtail          remove from tail

    Enjoy.

*****/

#ifndef SINGLIST_INCLUDE
#define SINGLIST_INCLUDE 1

#include <envstandards.h> /* MUST be first to configure */
#include <sys/types.h>
#include <limits.h>
#include <new>
#include <initializer_list>
#include <vsystem.h>
#include <localmisc.h>

/* external subroutines */

#if CF_DEBUGS
extern "C" int debugprintf(cchar *,...) ;
extern "C" int strlinelen(cchar *,cchar *,int) ;
#endif

/* local structures */

template <typename T>
class singlist ;

```

```

template <typename T>
class singlist_iter ;

template <typename T>
class singlist_node {
    singlist_node<T>          *next = NULL ;
    singlist_node<T>          *prev = NULL ;
    T                          val ;
    singlist_node(const singlist_node<T> &other) = delete ;
    singlist_node &operator = (const singlist_node<T> &other) = delete ;
    singlist_node(const T &av) : val(av) {
    } ;
    ~singlist_node() {
    } ;
    friend singlist<T> ;
    friend singlist_iter<T> ;
} ; /* end class (singlist_node) */

template <typename T>
class singlist_iter {
    singlist_node<T>          *n = NULL ;
    mutable T                  defval ;
public:
    singlist_iter() { } ;
    singlist_iter(singlist_node<T>* an) : n(an) { } ;
    singlist_iter(const singlist_iter<T> &it) {
        if (this != &it) {
            n = it.n ;
        }
    } ;
    singlist_iter(const singlist_iter<T> &&it) {
        if (this != &it) {
            n = it.n ;
        }
    } ;
    singlist_iter &operator = (singlist_iter<T> &it) {
        if (this != &it) {
            n = it.n ;
        }
        return (*this) ;
    } ;
    singlist_iter &operator = (singlist_iter<T> &&it) {
        if (this != &it) {
            n = it.n ;
        }
        return (*this) ;
    } ;
    singlist_iter &operator = (singlist_iter<T> *ip) {
        if (this != ip) {
            n = ip->n ;
        }
        return (*this) ;
    } ;
    ~singlist_iter() {
        n = NULL ;
    } ;
    friend bool operator == (const singlist_iter<T> &i1,
                             const singlist_iter<T> &i2) {
        return (i1.n == i2.n) ;
    } ;
    friend bool operator != (const singlist_iter<T> &i1,
                             const singlist_iter<T> &i2) {
        return (i1.n != i2.n) ;
    } ;
    T &operator * () const {

```

```

        T &rv = defval ;
        if (n != NULL) {
            rv = n->val ;
        }
        return rv ;
    } ;
    singlist_iter &operator ++ () { /* pre */
        if (n != NULL) {
            n = n->next ;
        }
        return (*this) ;
    } ;
    singlist_iter &operator ++ (int) { /* post */
        if (n != NULL) {
            n = n->next ;
        }
        return (*this) ;
    } ;
    singlist_iter &operator += (int inc) {
        if (n != NULL) {
            while ((n != NULL) && (inc-- > 0)) {
                n = n->next ;
            }
        }
        return (*this) ;
    } ;
    operator int() {
        return (n != NULL) ;
    } ;
    operator bool() {
        return (n != NULL) ;
    } ;
} ; /* end class (singlist_iter) */

template <typename T>
class singlist {
    singlist_node<T>          *head = NULL ;
    singlist_node<T>          *tail = NULL ;
    int                       c = 0 ;

public:
    typedef          singlist_iter<T> iterator ;
    typedef          T value_type ;
    singlist() = default ;
    singlist(const singlist<T> &al) {
        if (this != &al) {
            singlist_node<T>          *an = al.head ;
            if (head != NULL) clear() ;
            while (an != NULL) {
                instail(an->val) ;
                an = an->next ;
            }
        }
    } ;
    singlist(singlist<T> &&al) {
        if (this != &al) {
            if (head != NULL) clear() ;
            head = al.head ;
            tail = al.tail ;
            c = al.c ;
            al.head = NULL ;
            al.tail = NULL ;
            al.c = 0 ;
        }
    } ;
    singlist &operator = (const singlist<T> &al) {
        if (this != &al) {

```

```

        singlist_node<T>          *an = al.head ;
        if (head != NULL) clear() ;
        while (an != NULL) {
            instail(an->val) ;
            an = an->next ;
        }
    }
};

singlist &operator = (singlist<T> &&al) {
    if (this != &al) {
        if (head != NULL) clear() ;
        head = al.head ;
        tail = al.tail ;
        c = al.c ;
        al.head = NULL ;
        al.tail = NULL ;
        al.c = 0 ;
    }
};

singlist(const std::initializer_list<T> &list) {
    if (head != NULL) clear() ;
    for (const T &v : list) {
        instail(v) ;
    }
};

singlist &operator = (const std::initializer_list<T> &list) {
    if (head != NULL) clear() ;
    for (const T &v : list) {
        instail(v) ;
    }
    return (*this) ;
};

singlist &operator += (const std::initializer_list<T> &list) {
    for (const T &v : list) {
        instail(v) ;
    }
    return (*this) ;
};

singlist &operator += (const T v) {
    instail(v) ;
    return (*this) ;
};

~singlist() {
    singlist_node<T>          *nn, *n = head ;
    while (n != NULL) {
        nn = n->next ;
        delete n ;
        n = nn ;
    } /* end while */
    head = NULL ;
    tail = NULL ;
    c = 0 ;
};

int count() const {
    return c ;
};

int empty() const {
    return (c == 0) ;
};

operator int() const {
    return (c != 0) ;
};

operator bool() const {
    return (c != 0) ;
};

int clear() {

```

```

singlist_node<T>      *nn, *n = head ;
int      rc = c ;
while (n != NULL) {
    nn = n->next ;
    delete n ;
    n = nn ;
    rc += 1 ;
} /* end while */
head = NULL ;
tail = NULL ;
c = 0 ;
return rc ;
} ;
int instail(const T &v) {
singlist_node<T>      *nn = new(std::nothrow) singlist_node<T>(v) ;
int      rc = SR_NOMEM ; /* error indication */
if (nn != NULL) {
singlist_node<T>      *n = tail ;
if (n != NULL) {
    n->next = nn ;
} else {
    head = nn ;
}
tail = nn ;
rc = c++ ; /* return previous value */
} /* end if (allocation succeeded) */
return rc ;
} ;
int inshead(const T &v) {
singlist_node<T>      *nn = new(std::nothrow) singlist_node<T>(v) ;
int      rc = -1 ;
if (nn != NULL) {
singlist_node<T>      *n = head ;
if (n != NULL) {
    nn->next = n->next ;
} else {
    tail = nn ;
}
head = nn ;
rc = c++ ; /* return previous value */
} /* end if */
return rc ;
} ;
int insfront(const T &v) {
return inshead(v) ;
} ;
int insback(const T &v) {
return instail(v) ;
} ;
int ins(const T &v) {
return instail(v) ;
} ;
int add(const T &v) {
return instail(v) ;
} ;
int gethead(const T **rpp) const {
*rpp = (head != NULL) ? &head->val : NULL ;
return c ;
} ;
int gettail(const T **rpp) const {
*rpp = (tail != NULL) ? &tail->val : NULL ;
return c ;
} ;
int getfront(const T **rpp) const {
*rpp = (head != NULL) ? &head->val : NULL ;
return c ;
} ;

```

```

    } ;
int getback(const T **rpp) const {
    *rpp = (tail != NULL) ? &tail->val : NULL ;
    return c ;
} ;
int remhead(T *vp) {
    int rs = SR_EMPTY ;
    if (head != NULL) {
        singlist_node<T> *n = head ;
        if (vp != NULL) *vp = n->val ;
        head = n->next ;
        if (head == NULL) tail = NULL ;
        delete n ;
        rs = --c ;
    }
    return rs ;
} ;
int rem(T *vp) {
    return remhead(vp) ;
} ;
iterator begin() const {
    iterator it(head) ;
    return it ;
} ;
iterator end() const {
    iterator it ;
    return it ;
} ;
} ; /* end class (singlist) */

#endif /* SINGLIST_INCLUDE */

```