AI programming

```
// ai programming in c
//     tools:  list processing
//     Prabhas Chongstitvatana      27 Sept 2011

#include <ailib.h>

PRIVATE int cell[MAXCELL];
PRIVATE int freecell = ATOMTYPE;
PRIVATE char stringtable[MAXSTRING];
PRIVATE int freestring = 1;

/* available  functions
     head(a)
        tail(a)
        sethead(a,v)
        settail(a,v)
        isatom(a)          check a is atom
        islist(a)          check a is list
        name(str)          make name-atom
        number(n)          make number-atom
        cons(a,ls)         construct a list a:ls
        list(a)            make list from atom
        appenda(ls,a)      append list with atom
        clone(ls)          clone list
        print_list(ls)
*/

// a cell is a pair of integer
PRIVATE int newcell(int type, int value){
       int a;
       if(freecell > MAXCELL){
              fprintf(stderr,"out of cell memory");
              exit(1);
       }
       a = freecell;
       cell[a] = type;
       cell[a+1] = value;
       freecell += 2;
       return a;
}

// store string in the string buffer
// update freestring
PRIVATE int storestring(char *str){
       int idx;
       if(freestring > MAXSTRING){
              fprintf(stderr,"out of string memory");
              exit(1);
       }
       idx = freestring;
       strcpy(&stringtable[freestring],str);
       freestring += strlen(str) + 1;
       return idx;
}

char *get_string(int a){
       return &stringtable[tail(a)];
}

int head(int ls){
       return cell[ls];
}

int tail(int ls){
       return cell[ls+1];
```

```
}

void sethead(int ls, int v){
	cell[ls] = v;
}

void settail(int ls, int v){
	cell[ls+1] = v;
}

int isatom(int a){
	return head(a) < ATOMTYPE;
}

int islist(int a){
	return head(a) >= ATOMTYPE;
}

int name(char *str){
	return newcell(TYNAME, storestring(str));
}

int number(int n){
	return newcell(TYNUMBER, n);
}

// a is atom, ls is a list
int cons(int a, int ls){
	return newcell(a,ls);
}

int list(int a){
    return newcell(a, NIL);
}

int clone_atom(int a){
	return newcell(head(a),tail(a));
}

int clone(int ls){
	if( ls == NIL ) return NIL;
	if( isatom(head(ls)))
		return cons(clone_atom(head(ls)), clone(tail(ls)));
	else
		return cons(clone(head(ls)), clone(tail(ls)));
}

PRIVATE int gotoend(int ls){
	if( ls == NIL ) return NIL;
	if( tail(ls) == NIL ) return  ls;
	return gotoend(tail(ls));
}

// append atom a at the end of list
int appenda(int ls, int a){
	if( a == NIL ) return ls;
	settail(gotoend(ls), list(a));
	return ls;
}

PRIVATE void printstring(int a){
	printf("%c%s%c", 34,&stringtable[a],34);
}

PRIVATE void print_atom(int a){
	switch( head(a) ){
	case TYNAME: printstring(tail(a)); break;
	case TYNUMBER: printf("%d",tail(a)); break;
```

```
	}
}

void print_list(int a){
	if( a == NIL ) return;
	if( isatom(a) ){
		print_atom(a);
		printf(" ");
	}else{
		printf("(");
		while(a != NIL){
			print_list(head(a));
			a = tail(a);
		}
		printf(")");
	}
}
/*
void testlist(void){
	int a,b,c,d,e,f,g;
	a = number(1);
	print_list(a);
	nl();
	b = cons(a,NIL);
	print_list(b);
	nl();
	c = cons(number(1), cons(name("abc"), NIL));
	print_list(c);
	nl();
	d = appenda(c,number(4));
	print_list(d);
	nl();
	e = clone(d);
	print_list(e);
	nl();
	f = cons(d,e);
	g = clone(f);
	print_list(g);
	nl();
}
*/
//int main(void){
//	testlist();
//}
// basic example  4 Oct 2011

#include <ailib.h>

int fac(int n){
	if(n == 0) return 1;
	else return n * fac(n-1);
}

// using "result" as an accumulator
int fac2(int n, int result){
	if(n == 0) return result;
	else return fac2(n-1, n*result);
}

int length(int ls){
	if(ls == NIL) return 0;
	else return 1 + length(tail(ls));
}

void test_length(void){
	int a,b,c;
	a = cons(name("A"),cons(name("B"),NIL));
	b = clone(a);
```

```
	c = cons(a,b);
	print_list(c); nl();
	printf("%d\n",length(c));
}

// do length2 similar to fac2
// do fib

int rev2(int ls1, int ls2){
	if( ls1 == NIL ) return ls2;
	else return rev2(tail(ls1), cons(head(ls1),ls2));
}

int reverse(int ls){
	return rev2(ls,NIL);
}

void test_rev(void){
	int a,b,c;
	a = cons(name("A"),cons(name("B"),cons(name("C"),NIL)));
	print_list(a); nl();
	b = reverse(a);
	print_list(b); nl();
}

// append( (A B D), (C E) ) = (A B D C E)
// append( A, (B C) ) = no
// append( (A B C), NIL ) = (A B C)

// append( (A), NIL ) = (A)
// append( x, y ) =  if singleton(x) then return cons(head(x),y)

int append(int ls1, int ls2){
	if( ls1 == NIL ) return ls2;
	else return cons(head(ls1), append(tail(ls1),ls2));
}

void test_append(void){
	int a,b,c;
	a = cons(name("A"),cons(name("B"),cons(name("D"),NIL)));
	b = cons(name("C"),cons(name("E"),NIL));
	print_list(a); nl();
	print_list(b); nl();
	c = append(a,b);
	print_list(c); nl();
}

// with append now we can define "direct" reverse
//    it is not "efficient" as "reserve"
int rev3(int ls){
	if( ls == NIL ) return NIL;
	else return append(rev3(tail(ls)), cons(head(ls), NIL));
}

void test_rev3(void){
	int a,b,c;
	a = cons(name("A"),cons(name("B"),cons(name("C"),NIL)));
	print_list(a); nl();
	b = rev3(a);
	print_list(b);
}

// now try to do "rev3" without "append"

// rev( A B C D ) =  (D C B A)
//             = cons( D, (C B A) )
//  D = head(rev(tail(x)))
```

```
//  to get (C B A)
//     (C B A) = rev( A B C)
//             = rev( cons( A, (B C)))
//             going after tail(x)
//             we can get A from x
//             = rev(cons(head(x),(B C)))
//             = rev(cons(head(x),rev(C B)))
//             = rev(cons(head(x),rev(tail(D C B)))
//             = rev(cons(head(x),rev(tail(rev(tail(x))))))

//  rev(x) = cons(head(rev(tail(x)),
//                rev(cons(head(x),rev(tail(rev(tail(x)))))

int rev4(int ls){
       if( ls == NIL ) return NIL;
       else if( tail(ls) == NIL ) return ls;
       else return cons(head(rev4(tail(ls))),
                           rev4(cons(head(ls), rev4(tail(rev4(tail(ls)))))));
}

void test_rev4(void){
       int a,b,c;
       a = cons(name("A"),cons(name("B"),cons(name("C"),NIL)));
       print_list(a); nl();
       b = rev4(a);
       print_list(b);
}

int main(void){
//     printf("%d\n",fac2(6,1));
//     test_length();
//     test_rev();
//     test_append();
//     test_rev3();
       test_rev4();
}
// tree search    4 Oct 2011

#include <ailib.h>

#define eqs(str1,str2)    strcmp(str1,str2) == 0

int append(int ls1, int ls2){
       if( ls1 == NIL ) return ls2;
       else return cons(head(ls1), append(tail(ls1),ls2));
}

// association list = ( record record ...)
//    record = ( key attributes )

int chula;

int list2(char *str1, char *str2){
       return cons(name(str1),cons(name(str2),NIL));
}

int list3(char *str1, char *str2, char *str3){
       return cons(name(str1),cons(name(str2),cons(name(str3),NIL)));
}

int make_record(char *key, int att){
       return cons(name(key),list(att));
}

int make_chula(void){
       int a,b,c,d,e,f,g,h,i;
       a = list3("siam","rachaprasong","hualampong");
       b = make_record("chula",a);
```

```
       c = list3("mbk","paragon","natstadium");
       d = make_record("siam",c);
       e = list2("centralworld","pratunam");
       f = make_record("rachaprasong",e);
       g = list(name("yaowarad"));
       h = make_record("hualampong",g);
       return cons(b,cons(d,cons(f,cons(h,NIL))));
}

extern char *get_string(int a);

int atomeq(int x, int y){
       if( ! isatom(x) ) return 0;
       if( ! isatom(y) ) return 0;
       if( head(x) != head(y) ) return 0;
       if( head(x) == TYNUMBER ) return tail(x) == tail(y);
    return eqs(get_string(x),get_string(y));
}

void test_atomeq(void){
       int a, b, c, d;
       a = name("A");
       b = name("A");
       c = number(2);
       d = number(2);
//     printf("%s\n",get_string(a));
       printf("%d\n",atomeq(a,b));
       printf("%d\n",atomeq(c,d));
}

int search_asso(int key, int alist){
       if( atomeq(key, head(head(alist)))) return head(tail(head(alist)));
       else return search_asso(key, tail(alist));
}

int main(void){
       int a;
//     a = list3("A","B","C");
//     print_list(a); nl();
       chula = make_chula();
       print_list(chula); nl();
//     test_atomeq();
       a = search_asso(name("chula"),chula);
       print_list(a); nl();
}
```