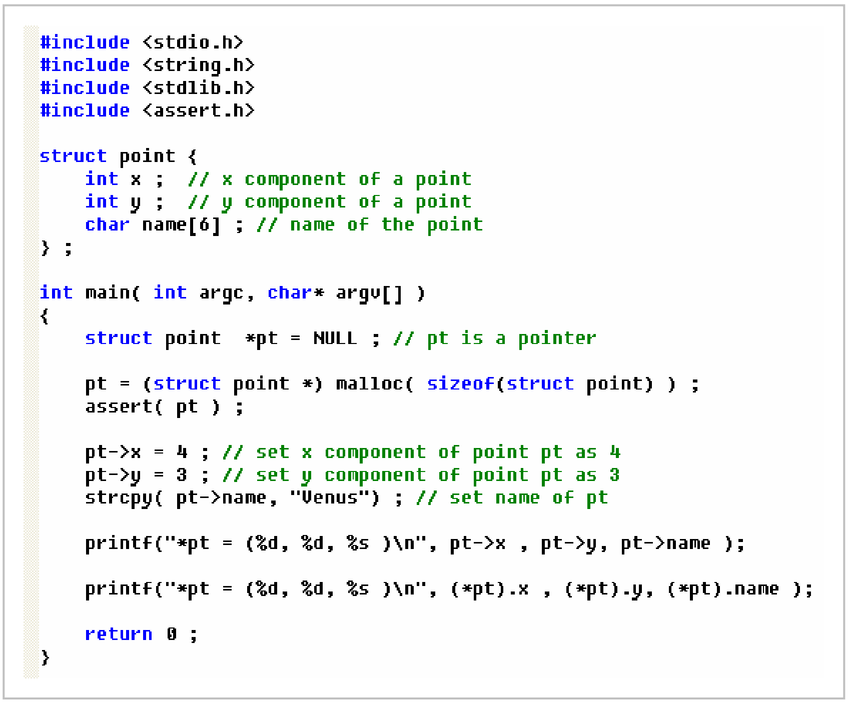
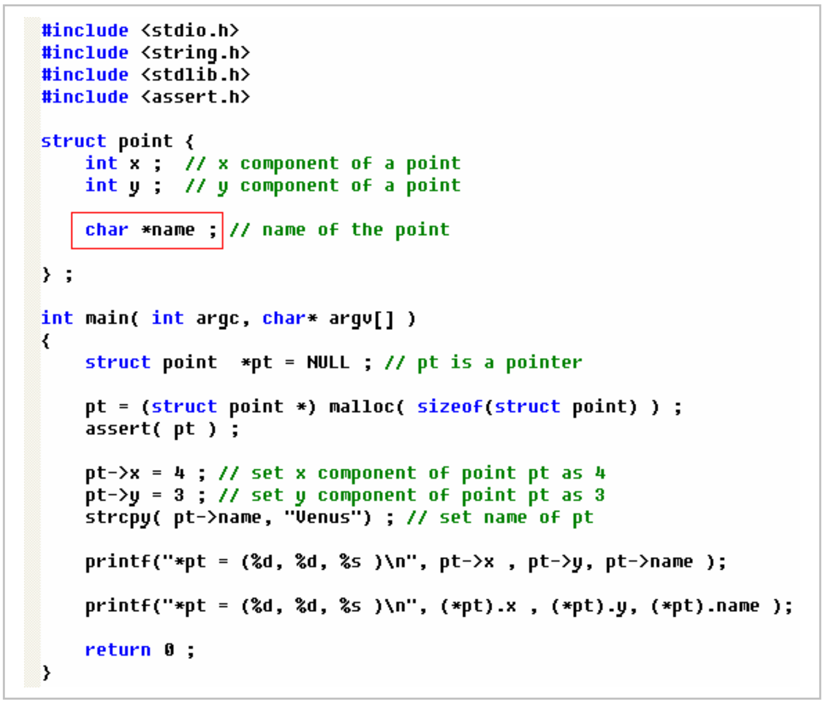
2008 summer course, C-language   Homework 6

Exercise 1(heterogeneous data aggregation): implement codes in Figure 1 and check its address of each variable, in the code we use string copy ***strcpy*** to setup name of point as Venus. This operation is not safe, for example, consider code in Figure 2, we modify struct point such that name field is defined as character pointer, not character array, what happens when you execute code in Figure 2? Can you explain the error? Write a correct one.



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

struct point {
    int x ;  // x component of a point
    int y ;  // y component of a point
    char name[6] ; // name of the point
} ;

int main( int argc, char* argv[] )
{
    struct point  *pt = NULL ; // pt is a pointer

    pt = (struct point *) malloc( sizeof(struct point) ) ;
    assert( pt ) ;

    pt->x = 4 ; // set x component of point pt as 4
    pt->y = 3 ; // set y component of point pt as 3
    strcpy( pt->name, "Venus") ; // set name of pt

    printf("*pt = (%d, %d, %s )\n", pt->x , pt->y, pt->name );

    printf("*pt = (%d, %d, %s )\n", (*pt).x , (*pt).y, (*pt).name );

    return 0 ;
}
```

Figure 1: structure point and its manipulation.



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>

struct point {
    int x ;  // x component of a point
    int y ;  // y component of a point

    char *name ; // name of the point

} ;

int main( int argc, char* argv[] )
{
    struct point  *pt = NULL ; // pt is a pointer

    pt = (struct point *) malloc( sizeof(struct point) ) ;
    assert( pt ) ;

    pt->x = 4 ; // set x component of point pt as 4
    pt->y = 3 ; // set y component of point pt as 3
    strcpy( pt->name, "Venus") ; // set name of pt

    printf("*pt = (%d, %d, %s )\n", pt->x , pt->y, pt->name );

    printf("*pt = (%d, %d, %s )\n", (*pt).x , (*pt).y, (*pt).name );

    return 0 ;
}
```

Figure 2: change name field in structure point as character pointer, not a character array.

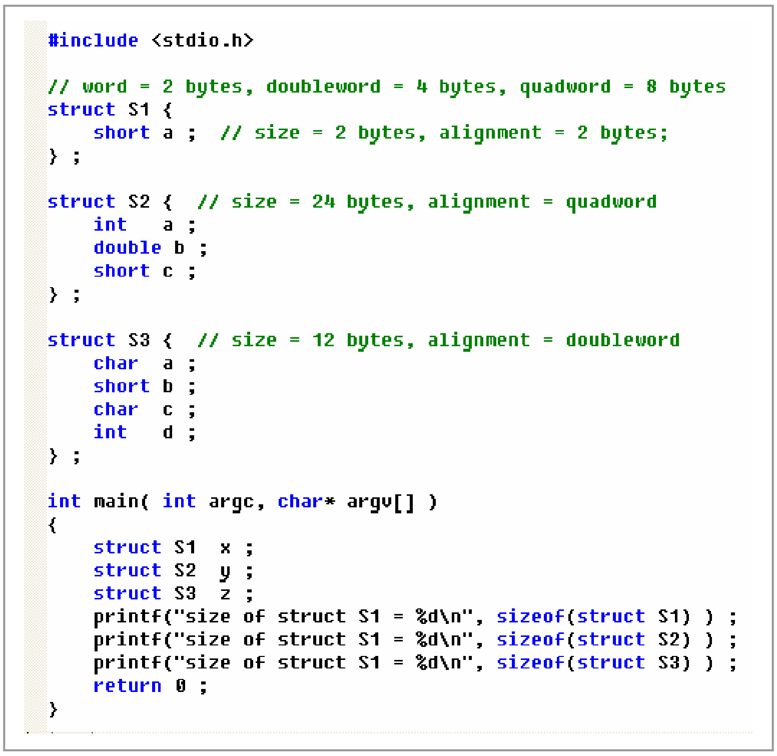
Exercise 2(padding and alignment): In the course, we introduce padding technique of compiler, and show you some alignment of basic data type in **Table 1**.

(1) write codes in Figure 3 (we talk about in the course) and show the alignments by debugger (the graph of alignment has been shown in the course)

(2) Can you explain the alignment?

Table 1: suggested alignment for the scalar members of structures

C data type	alignment	C data type	Alignment
char	byte	short	Word (2 bytes)
int	doubleword (4 bytes)	double	quadword (8 bytes)



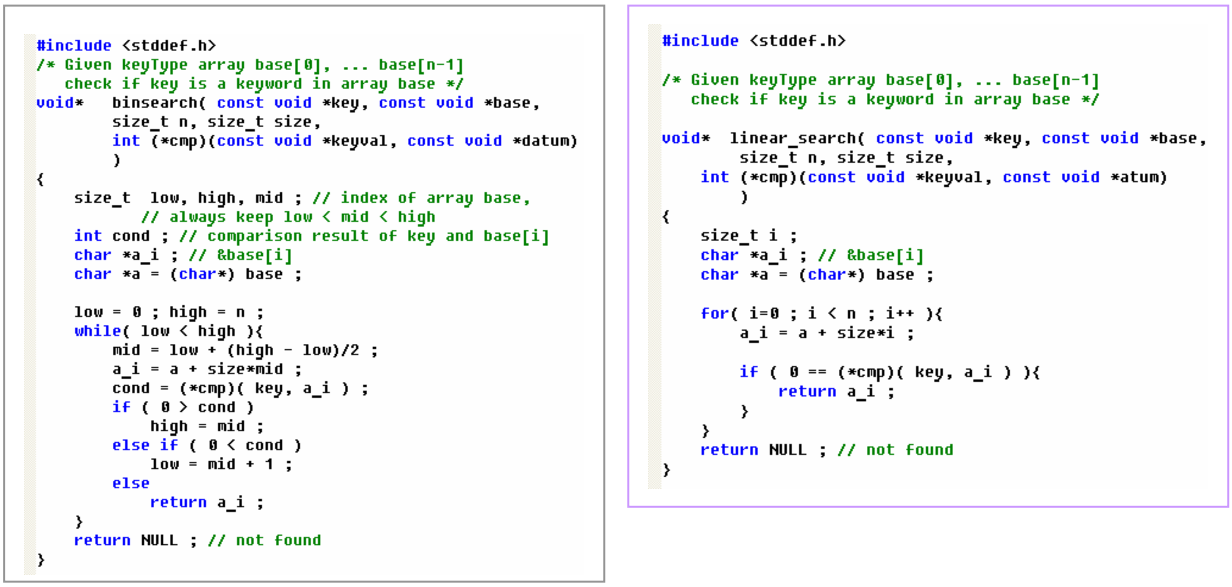
```
#include <stdio.h>

// word = 2 bytes, doubleword = 4 bytes, quadword = 8 bytes
struct S1 {
    short a ;  // size = 2 bytes, alignment = 2 bytes;
} ;

struct S2 {  // size = 24 bytes, alignment = quadword
    int    a ;
    double b ;
    short c ;
} ;

struct S3 {  // size = 12 bytes, alignment = doubleword
    char   a ;
    short b ;
    char   c ;
    int    d ;
} ;

int main( int argc, char* argv[] )
{
    struct S1  x ;
    struct S2  y ;
    struct S3  z ;
    printf("size of struct S1 = %d\n", sizeof(struct S1) ) ;
    printf("size of struct S1 = %d\n", sizeof(struct S2) ) ;
    printf("size of struct S1 = %d\n", sizeof(struct S3) ) ;
    return 0 ;
}
```

Figure 3: padding example

Exercise 3(binary search): in the course, we introduce framework of linear search and binary search, also we use technique of function pointer to implement these two algorithms, see Figure 4.

(1) implement these two algorithms and test these two algorithms on structure array ***keytab*** we talk about in the course. Which algorithm is efficient? Explain your reason

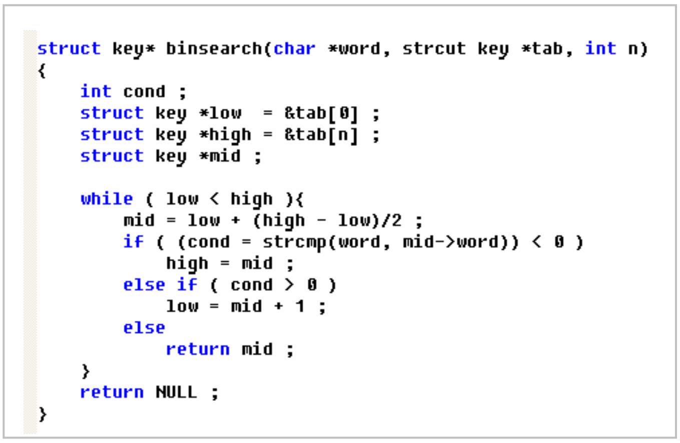
(2) in page 137 of textbook, the author provide a binary search as Figure 5, verify that this algorithm does work on your test in (1). What is pros and cons (優缺點) of this binary search?



```
#include <stddef.h>
/* Given keyType array base[0], ... base[n-1]
   check if key is a keyword in array base */
void*   binsearch( const void *key, const void *base,
        size_t n, size_t size,
        int (*cmp)(const void *keyval, const void *datum)
        )
{
    size_t  low, high, mid ; // index of array base,
           // always keep low < mid < high
    int cond ; // comparison result of key and base[i]
    char *a_i ; // &base[i]
    char *a = (char*) base ;

    low = 0 ; high = n ;
    while( low < high ){
        mid = low + (high - low)/2 ;
        a_i = a + size*mid ;
        cond = (*cmp)( key, a_i ) ;
        if ( 0 > cond )
            high = mid ;
        else if ( 0 < cond )
            low = mid + 1 ;
        else
            return a_i ;
    }
    return NULL ; // not found
}
```

```
#include <stddef.h>

/* Given keyType array base[0], ... base[n-1]
   check if key is a keyword in array base */

void*  linear_search( const void *key, const void *base,
        size_t n, size_t size,
    int (*cmp)(const void *keyval, const void *atum)
        )
{
    size_t i ;
    char *a_i ; // &base[i]
    char *a = (char*) base ;

    for( i=0 ; i < n ; i++ ){
        a_i = a + size*i ;

        if ( 0 == (*cmp)( key, a_i ) ){
            return a_i ;
        }
    }
    return NULL ; // not found
}
```

Figure 4: framework of binary search (left panel) and linear search (right panel)



```
struct key* binsearch(char *word, strcut key *tab, int n)
{
    int cond ;
    struct key *low  = &tab[0] ;
    struct key *high = &tab[n] ;
    struct key *mid ;

    while ( low < high ){
        mid = low + (high - low)/2 ;
        if ( (cond = strcmp(word, mid->word)) < 0 )
            high = mid ;
        else if ( cond > 0 )
            low = mid + 1 ;
        else
            return mid ;
    }
    return NULL ;
}
```

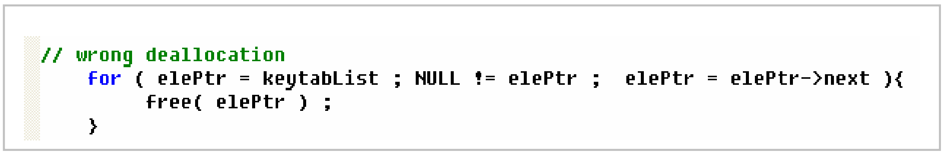
Figure 5: binary search in page 137 of textbook.

Exercise 4(linked list) : In the course we say linked list is a discontinuous array and demonstrate this via code in Figure 6.

(1) Implement code in Figure 6 and run it on Linux machine, does this discontinuous nature hold?

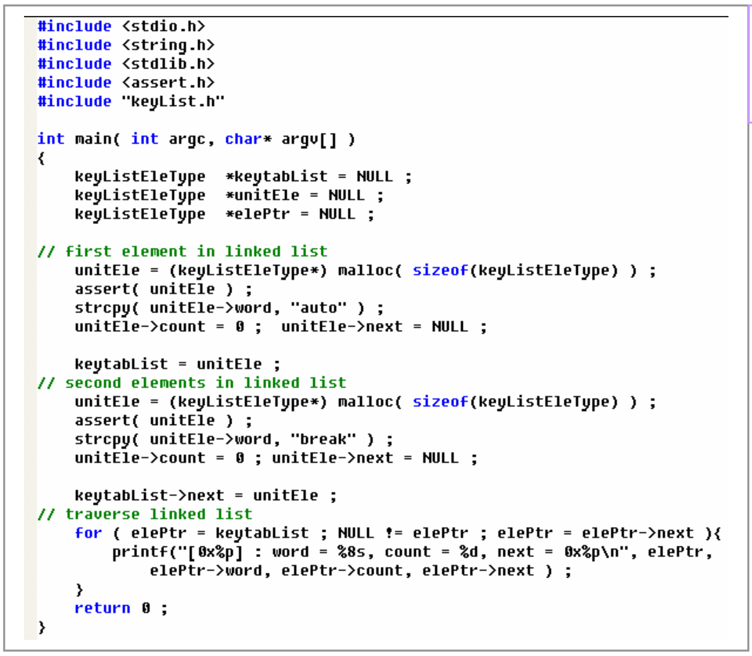
(2) In Figure 6, we only create a linked list of two elements (this is just first two elements of structure array ***keytab***), try to build a liked list corresponding to structure array ***keytab.***

(3) When we de-allocate linked list, wrong procedures would cause program crash,

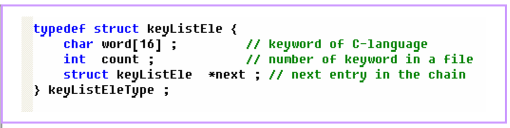


```
// wrong deallocation
    for ( elePtr = keytabList ; NULL != elePtr ;  elePtr = elePtr->next ){
        free( elePtr ) ;
    }
```

Explain why above code does not work and run above code in Linux machine, what is error message?



```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include "keyList.h"

int main( int argc, char* argv[] )
{
    keyListEleType  *keytabList = NULL ;
    keyListEleType  *unitEle = NULL ;
    keyListEleType  *elePtr = NULL ;

// first element in linked list
    unitEle = (keyListEleType*) malloc( sizeof(keyListEleType) ) ;
    assert( unitEle ) ;
    strcpy( unitEle->word, "auto" ) ;
    unitEle->count = 0 ;  unitEle->next = NULL ;

    keytabList = unitEle ;
// second elements in linked list
    unitEle = (keyListEleType*) malloc( sizeof(keyListEleType) ) ;
    assert( unitEle ) ;
    strcpy( unitEle->word, "break" ) ;
    unitEle->count = 0 ; unitEle->next = NULL ;

    keytabList->next = unitEle ;
// traverse linked list
    for ( elePtr = keytabList ; NULL != elePtr ; elePtr = elePtr->next ){
        printf("[0x%p] : word = %8s, count = %d, next = 0x%p\n", elePtr,
            elePtr->word, elePtr->count, elePtr->next ) ;
    }
    return 0 ;
}
```



```c
typedef struct keyListEle {
    char word[16] ;          // keyword of C-language
    int  count ;             // number of keyword in a file
    struct keyListEle  *next ; // next entry in the chain
} keyListEleType ;
```

Figure 6: construct linked list and traverse it.

Exercise 5(hash table): read section 6.6 in page 143 of textbook and write codes in the book.