Massachusetts Institute of Technology
Department of Electrical Engineering and Computer Science

6.087: Practical Programming in C

IAP 2010

**Problem Set 6 – Solutions**
Part 2: Function pointers, hash table

**Out:** Thursday, January 21, 2010. **Due:** Friday, January 22, 2010.

### Problem 6.1

In this problem, we will use and create function that utilize function pointers. The file 'callback.c' contains an array of records consisting of a fictitious class of celebrities. Each record consists of the firstname, lastname and age of the student. Write code to do the following:

- Sort the records based on first name. To achieve this, you will be using the `qsort()` function provided by the standard library: `void qsort(void* arr,int num,int size,int (*fp)(void* pa,void*pb))`. The function takes a pointer to the start of the array 'arr', the number of elements 'num' and size of each element. In addition it takes a function pointer 'fp' that takes two arguments. The function fp is used to compare two elements within the array. Similar to `strcmp()`, it is required to return a negative quantity,zero or a positive quantity dependeing on whether element pointed to by 'pa' is "less" than, equal to or "greater" the element pointed to by 'pb'. You are required to write the appropriate *callback* function.
- Now sort the records based on last name. Write the appropriate *callback function*.
- The function `void apply(...)` iterates through the elements of the array calling a function for each element of the array. Write a function `isolder()` that prints the record if the age of the student is greater 20 and does nothing otherwise.

Answer: Here's one possible implementation:

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

#define MAX_STUDENTS 10
struct student
{
  char fname[100];
  char lname[100];
  int  year;
  int  age;
};

struct student class[]={
  "Sean","Penn",2,21,
  "Sean","Connery",4,25,
  "Angelina","Jolie",3,22,
  "Meryl","Streep",4,29,
  "Robin","Williams",3,32,
  "Bill","Gates",3,17,
  "Jodie","Foster",4,25,
  "John","Travolta",1,17,
  "Isaac","Newton",2,19,
  "Sarah","Palin",2,19
};


/*
  @function compare_first_name
  @desc     compares first name of two records.
*/
int compare_first_name(const void* a,const void* b)
{
  struct student* sa=(struct student*)a;
  struct student* sb=(struct student*)b;
  return strcmp(sa->fname,sb->fname);
}

/*
  @function compare_lname_name
  @desc     compares last name of two records.
*/
int compare_last_name(const void* a,const void* b)
{
  struct student* sa=(struct student*)a;
  struct student* sb=(struct student*)b;
  return strcmp(sa->lname,sb->lname);
}

/*!
  @function apply
  @desc     applies
 */
void apply(struct student* sarr,int nrec,void (*fp)(void* prec,void* arg),void* arg)
{
  int i=0;
  for(i=0;i<nrec;i++)
```

```
    {
      /*callback*/
      fp(&sarr[i],arg);
    }
}

/*
  @function printrec
  @desc     prints student record
*/
void printrec(void* prec,void* arg)
{
  struct student* pstud=(struct student*)prec;
  printf("%-20s %-20s %2d %2d\n",pstud->fname,pstud->lname,pstud->year,pstud->age);
}


/*
  @function isolder
  @desc     prints student record
*/
void isolder(void* prec,void* arg)
{
  int* age=(int*)arg;
  struct student* pstud=(struct student*)prec;
  if(pstud->age < *age)
    return; /*do nothin*/
  else
    printf("%-20s %-20s %2d %2d\n",pstud->fname,pstud->lname,pstud->year,pstud->age);
}

int main()
{
  int nstudents=sizeof(class)/sizeof(struct student);
  int age;

  puts("Raw records:");
  puts("------------------------------");
  apply(class,nstudents,printrec,NULL);

  /*sort based on first name*/
  puts("Sorted by first name:");
  puts("------------------------------");
  qsort(class,nstudents,sizeof(struct student),compare_first_name);
  apply(class,nstudents,printrec,NULL);

  /*sort based on last name*/
  puts("Sorted by last name:");
  puts("------------------------------");
  qsort(class,nstudents,sizeof(struct student),compare_last_name);
  apply(class,nstudents,printrec,NULL);

  /*print people older than 20*/
  puts("People older than 20:");
  puts("------------------------------");
  age=20;
  apply(class,nstudents,isolder,&age);
  return 0;
}
```

**Problem 6.2**

A useful data structure for doing lookups is a hash table. In this problem, you will be implementing a hash table with chaining to store the frequency of words in a file. The hash table is implemented as an array of linked lists. The hash function specifies the index of the linked list to follow for a given word. The word can be found by following this linked list. Optionally, the word can be appended to this linked list. You will require the code file 'hash.c' and data file 'book.txt' for this problem. You are required to do the following

- The function `lookup()` returns a pointer to the record having the required string. If not found it returns NULL or optionally creates a new record at the correct location. Please complete the rest of the code.
- Complete the function `cleartable()` to reclaim memory. Make sure each call to `malloc()` is matched with a `free()`

Answer: one possible implementation is shown below:

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


#define MAX_BUCKETS 1000
#define MULTIPLIER 31
#define MAX_LEN     100

struct wordrec
{
  char* word;
  unsigned long count;
  struct wordrec* next;
};

struct wordrec* walloc(const char* str)
{
  struct wordrec* p=(struct wordrec*)malloc(sizeof(struct wordrec));
  if(p!=NULL)
  {
      p->count=0;
      p->word=strdup(str); /*creates a duplicate*/
      p->next=NULL;
  }
  return p;
}

/*hash bucket*/
struct wordrec* table[MAX_LEN];

/*
  @function hashstring
  @desc     produces hash code for a string
            multipliers 31,35 have been found to work well
*/
unsigned long hashstring(const char* str)
{
  unsigned long hash=0;
  while(*str)
    {
      hash= hash*MULTIPLIER+*str;
      str++;
    }
  return hash%MAX_BUCKETS;
}


/*
  @function lookup
  @desc     returns a pointer to the word or creates
            it if required
*/
struct wordrec*  lookup(const char* str,int create)
{
  unsigned long hash=hashstring(str);
  struct wordrec* wp=table[hash];
```

```
  struct wordrec* curr=NULL;
  for(curr=wp;curr!=NULL ;curr=curr->next)
    if(strcmp(curr->word,str)==0) /*found*/
    {
  return curr;
    }
   /*not found*/
  if(create)
    {
      curr=(struct wordrec*)malloc(sizeof(struct wordrec));
      curr->word=strdup(str);
      curr->count=0;
      /*add to front*/
      curr->next=table[hash];
      table[hash]=curr;
    }
  return curr;
}

/*
  @function cleartable()
  @desc     reclaims memory
*/
void cleartable()
{
  struct wordrec* wp=NULL,*p=NULL;
  int i=0;
  for(i=0;i<MAX_BUCKETS;i++)
   {
      wp=table[i];
      while(wp)
      {
    p=wp;
    wp=wp->next;

    free(p->word);
    free(p);
       }
   }
}

int main(int argc,char* argv[])
{
  FILE* fp=fopen("book.txt","r");
  char  word[1024]; /*big enough*/
  struct wordrec* wp=NULL;
  int i=0;

  memset(table,0,sizeof(table));
  /*read from input*/
  while(1)
  {
    if(fscanf(fp,"%s",word)!=1)
      break;
    wp=lookup(word,1); /*create if doesn't exist*/
    wp->count++;
  }
  fclose(fp);
```

```
  /*
    print all words have frequency>100
   */
  for(i=0;i<MAX_BUCKETS;i++)
    {
      for(wp=table[i];wp!=NULL;wp=wp->next)
  {
    if(wp->count>1000)
      {
        printf("%s-->%ld\n",wp->word,wp->count);
      }
  }
    }
  cleartable();
  return 0;
}
```

6.087 Practical Programming in C

January (IAP) 2010