School of CSE,
KLE TECH
2017-18

8. Storage Management

Data Structures and Algorithms

17ECSC204

Prakash B Hegade

Variable Length Records

- Different record in the file have different size
- Computer does not know exact location of record so slow access
- Fast transferring as it is small in size

Sample Program:

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

void add_customer()
{
   FILE * fp;
   char filename[30] = "customer.txt";
   int registration_number;
   char name[20];
   int age;
   char city[20];
   char delimiter = '|';

   fp = fopen(filename, "a");
   if(fp == NULL) {
      printf("Cannot open the file\n");
      exit(0);
   }

   printf("Enter the following details: Reg Num, name, Age, City\n\n");
   scanf("%d %s %d %s", &registration_number, name, &age, city);

   fprintf(fp, "%d %s %d %s %c ", registration_number, name, age,city, delimiter);
   fclose(fp);
}

void display()
{
   FILE * fp2;
   char filename[30] = "customer.txt";

   int registration_number;
   char name[20];
   int age;
   char city[20];
   char delimiter;

   fp2 = fopen(filename, "r");
```

```
   if(fp2 == NULL) {
      printf("Cannot open the file\n");
      exit(0);
   }

  while(1) {
     if(feof(fp2))
      break;
     fscanf(fp2, "%d %s %d %s %c ", &registration_number, name, &age, city, &delimiter);
     printf("%d %s %d %s\n", registration_number, name, age, city);
  }
   fclose(fp2);
}

int main()
{
   int choice = 0;
   while(1)
   {
      printf("Menu\n");
      printf("1- Register a new customer\n2-Display All customers\n3-Exit\n\n");
      printf("Enter your choice\n");
      scanf("%d", &choice);

      switch(choice)
      {
         case 1: printf("New customer will be added to file\n");
               add_customer();
               printf("Customer added successfully\n");
               break;

         case 2: display();
               break;

         case 3: exit(0);
      }
   }
   return 0;
}
```

Fixed Length Records

- Every record in the file has exactly same size (in byte)
- Computer knows exact location of records so easy access
- Slow in transferring the records it has large size.

Sample Program:

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

void add_customer()
{
   FILE * fp;
   char filename[30] = "customer.txt";
   int registration_number;
   char name[20];
   int age;
   char city[20];
   int count = 0;
   int record_length = 50;
   int remain = 0;
   int space_count = 3;
   int i;

   fp = fopen(filename, "a");
   if(fp == NULL) {
      printf("Cannot open the file\n");
      exit(0);
   }

   printf("Enter the following details: Reg Num, name, Age, City\n\n");
   scanf("%d %s %d %s", &registration_number, name, &age, city);

   int trn, tage;
   trn = registration_number;
   tage = age;

   // Get the number of bytes of integer data
   while(trn != 0) {
      trn /= 10;
      count++;
   }

   while(tage != 0) {
      tage /= 10;
      count++;
   }

   // Add the count from strings and spaces
   count = count + strlen(name) + strlen(city);
   count = count + space_count;
```

```
    // Calculate the remain, needed to pad the extra bytes into the file
    remain = record_length - count;

    // Write the data into the file
    fprintf(fp, "%d %s %d %s ", registration_number, name, age,city);

    // Pad the remain with a special character not occurring in the file – say '#'
    for (i = 0; i < remain; i++)
        putc('#', fp);

    fclose(fp);
}

void display()
{
    FILE * fp2;
    char filename[30] = "customer.txt";

    int registration_number;
    char name[20];
    int age;
    char city[20];
    int count = 0;

    fp2 = fopen(filename, "r");
    if(fp2 == NULL) {
        printf("Cannot open the file\n");
        exit(0);
    }

    char ch;
    while((ch=getc(fp2)) != EOF)
    {
        if (ch != '#'){
            printf("%c", ch);
        }
        count ++;
        if(count  == 50) {
            printf("\n");
            count  = 0;
        }
    }

    fclose(fp2);
}
```

```
int main()
{
   int choice = 0;

   while(1)
   {
      printf("Menu\n");
      printf("1- Register a new customer\n2-Display All customers\n3-Exit\n\n");
      printf("Enter your choice\n");
      scanf("%d", &choice);

      switch(choice)
      {
         case 1: printf("New customer will be added to file\n");
               add_customer();
               printf("Customer added successfully\n");
               break;

         case 2: display();
               break;

         case 3: exit(0);
      }
   }
   return 0;
}
```

Note:
For differences between Fixed and Variable length Records, Refer class notes.

Bit Maps

(Referenced from Wikipedia)
In computing, a bitmap is a mapping from some domain (for example, a range of integers) to bits (values which are zeros and ones). It is also called a bit array or bitmap index.

A bitmap is a type of memory organization or image file format used to store digital images. The term bitmap comes from the computer programming terminology, meaning just a map of bits, a spatially mapped array of bits.

BMP is a bitmap file format. Similarly, most other image file formats, such as JPEG, TIFF, PNG, and GIF, also store bitmap images (as opposed to vector graphics), but they are not usually referred to as bitmaps, since they use compressed formats internally.

Dynamic Memory Allocation API's

malloc – allocates and reserves a block of memory, specified in bytes and returns a pointer to the first byte of allocated space.
Example: malloc(size)
char *str;
str = (char *)malloc(10);

calloc – allocates multiple block of same size, initializes all locations to zero and returns a pointer to the first byte of allocated space
Example: calloc(n, size)
char* str=NULL;
str = (char *)calloc(10, sizeof(char));

realloc – used to alter the previously allocated space which is allocated by malloc or calloc
Example:
char *str;
str = (char *)malloc(10);
str = (char *) realloc(str, 40);

free –release the memory space that had been allocated by memory allocation functions
Example: char *str;
str = (char *)malloc(10);
free(str);

Static and Dynamic Memory Allocation

Differences between two memory allocation Schemes:

Si. No	Static Memory Allocation	Dynamic Memory Allocation
1	Memory allocation happens at compile time	Memory allocation happens at Run time
2	It applies to global variables, file scope variables, and variables qualified with static defined inside functions	It applies to variables that allocate memory during runtime using memory allocation API's
3	The size is fixed when the program is created	Programmer can control the exact size and the lifetime of the memory locations
4	Memory allocated at compile time in stack or other data segments	Memory is allocated during run-time in heap
5.	This is used when the size of memory is variable and is known only during run-time	This is used when the size of memory is static/constant and is known during compile-time

6.	The compiler allocates the required memory space for a declared variable	It uses functions such as malloc( ) or calloc( ) to get memory dynamically
7.	Memory is allocated before the execution of the program begins	Memory is allocated during the execution of the program
8.	Static allocation will be much faster. Static allocation can happen at global scope, and on the stack. So, Faster execution than Dynamic	Dynamic memory must be allocated from a heap, and even in the best case most allocations will take time that scales more than linear with each allocation. So, Slower execution than static
9.	More memory Space required	Less Memory space required

Files and Linked List

Program to load the integer data from file and add them at the end of singly linked list. The file named 'numbers.txt' will hold the integer numbers separated by space.

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
   int data;
   struct node *next;
};
typedef struct node NODE;

NODE * insert_at_end(NODE * start, int data)
{
   NODE * newnode, * nextnode;
   newnode = (NODE *)malloc(sizeof(NODE));
   if(newnode == NULL) {
      printf("Memory Allocation Failed\n");
      return start;
   }
   newnode->data = data;
   newnode->next = NULL;

   if(start == NULL)
      start = newnode;
   else {
      nextnode = start;
      while(nextnode->next != NULL)
         nextnode = nextnode->next;
      nextnode->next = newnode;
   }
   return start;
}
```

```
void display(NODE * start)
{
   printf("The list read from file is...\n");
   while(start!=NULL) {
      printf("%d\t", start->data);
      start = start->next;
   }
}

int main()
{
   FILE *fp;
   int num;
   NODE *start = NULL;
   fp = fopen("numbers.txt", "r");
   if(fp == NULL) {
      printf("Unable to open file\n");
      return 0;
   }

   while(1) {
      if(feof(fp))
         break;

      fscanf(fp, "%d ", &num);
      start = insert_at_end(start, num);
   }
   fclose(fp);
   display(start);
   return 0;
}
```

~*~*~*~*~*~