Åbo Akademi

# Programmering i C/C++

Räkneövning 1

Torsdag 6.11.2008

# Uppgift 1

- Initialisera en char räcka, dvs reservera minne för arrayen från stacken
  t.ex. char array[256];
- Läs in en text som du skriver in i konsolen och spara den i char räckan
  TIPS! Studera följande:
  - stdin
  - gets()
  - fgets()
- Skriv ut texten i räckan till konsolen genom att använda: **printf()**
- Kom ihåg att inkludera ändamålsenliga bibliotek!

```
/* Räkneövning 1, Uppgift 1*/
#include<stdio.h>
#define MAX_LEN 254

int main (int argc, char *argv[])
{
        char text[MAX_LEN];
        char *ptr = NULL;

        ptr = fgets(/*Ange rätta parametrar*/);

        if (ptr == NULL)
        {
                printf(”An error occured ”);
                exit (EXIT_FAILURE);
        }
        else
        {
                /*printa ut det du skrivit in från stdin*/
        }
        return (0);
}
```

# Uppgift 2

- Definiera MAX_LENGHT med **#define**
- **char *argv[1]** innehåller (pekar på) filnamnet som anges i command prompten medan **int argc** anger antalet inputparamterar
- Deklarera och initialisera FILE pekare med:
  **FILE *p = NULL;**
- Använd **fopen()** för att öppna filström
- Kom ihåg att skriva ”säker” kod, dvs. se till att filerna öppnas korrekt genom att använda IF-ELSE satser och perror
- Funktionerna **fgets(), strncpy()** och **fprintf()** kommer till nytta i denna övning
- Skriv en skild funktion som utför skrivandet till fil
- Kom ihåg att stänga filströmmar!!! T.ex. **fclose (p);**

```
/* Räkneövning 1, Uppgift 2*/
#includera biblioteket för input/output
#includera biblioteket för stränghantering
#includera biblioteket för användning av exit(), atoi, m.fl.

/*Structen*/
struct address {
....
};
/*Prototyper*/
void printText(struct *, FILE *);
int main( int argv, char *argc[]) {
        struct address ADDRESS;
        FILE *fileDB = NULL;
        fileDB = fopen (argv[1], "w");
        if (fileDB == NULL) {
                /* Avbryt programmet ifall fileDB pekar på NULL*/
        }else {
                /* Mata in uppgifterna till structen*/
        }
        /*Anropa funktionen printText med structen och en file stream som param.*/
        printText(&ADDRESS, fileDB);
        /* Kom ihåg att stänga filströmmen! */
        fclose (fileDB);
        return 0;
}

void printText( struct *adr, FILE *file)
{
        if ( file == NULL)              // Checka att file inte pekar på NULL
        {
                return -1;
        }else {
                /*Skriv till stdout eller till fil genom att t.ex skriva:*/
                fprintf(file, "Name: %s %s\n", adr->name, adr->surname);
                ........
        }
}
```

# Uppgift 3

- Denna uppgift baserar sig långt på uppgift 2. Det nya i den här uppgiften är att skapa en länkad lista samt arbeta med dynamisk minnesallokering.
- Dynamisk minnesallokering:
  - **malloc()**
  - **calloc()**
- I denna uppgift är det också väsentligt att frigöra allokerat minne. Detta görs med **free()**
- Skriv helst en skild funktion för följande uppgifter:
  - lägga till nya element (dvs, adresser) till listan
  - Läsa data
  - Skriva ut adresslistan till standard output (stdout) och fil
  - Tömma listan
- **Kom ihåg att stänga filströmmen med fclose() !!**

```
/* Räkneövning 1, Uppgift 2*/
#includera biblioteket för input/output
#includera biblioteket för stränghantering
#includera biblioteket för användning av exit(), atoi, m.fl.
/*Structen*/
struct address {};
struct nod {};
/* Prototyper */
int readData(struct address *);
int newAddressToList();
int writeAddress (FILE * );
int emptyList ();
/* Global pekare */
struct nod * listHead = NULL;

int main( int argv, char *argc[]) {
        struct address ADDRESS;
        FILE *fileDB = NULL;
        if (fileDB pekar på null) {
                /* returnera felmeddelande och stäng programmet */
        } else {
                /* Lägg till minst 3 element till den länkade listan */
                int i;
                for (i=0; i < 3; i++) { newAddressToList();  }

                /* Skriv sedan listan till stdout och filen */
                 writeAddress (fileDB);
                 writeAddress (stdout);

        /* Töm tillslut  den länkade listan och stäng filströmmen */
        emptyList();
        fclose (fileDB);
        return 0;
}
int readData(struct address * addr) {...}
int newAddressToList() { ... }
int writeAddress (FILE * fil) { ...}
int emptyList () {  ... }
```

- I functionerna readData(), newAddressToList(), writeAddress () och emptyList() löns det att deklarera en temporär nod som pekar på listHead enligt följande:
  struct nod *temp = NULL;
  temp = listHead;
- Använd sedan den temporära noden för att stega egenom den länkade listan.
- I newAddressToList() finns det två fall: antingen är listan tom eller så finns det element i den. Ifall listHead är tom börjar man alltså med att allokera minne för ett element i listan:

  listHead = (struct nod *) malloc( sizeof(struct nod) );

  Annars så sätter man temp = listHead och stegar genom listan tills temp->next == NULL. Därefter utför man en minnesallokering enligt följande:

  temp->next = (struct nod *) malloc( sizeof(struct nod) );
  temp = temp->next;

- När minnesallokeringen är utförd bör man komma ihåg att sätta det sista elementet i listan till NULL, dvs temp->next = NULL.
- Sedan är det dags för att mata in en ny adress genom att anropa readData(&temp->data); Det är denna funktion som användes I förgående uppgift.

- I functionen emptyList() löns det att till en början skapa två noder:
  struct nod *temp = listHead;
  struct nod *toFree = NULL;

- Tanken är alltså att stegvis uppdatera toFree med temp tills temp == NULL, dvs

  toFree = temp;
  temp = temp->next;
  free(toFree);