F12 - repetition av F9-F11

Programmeringsteknik med C och Matlab, 7,5 hp

Niclas Börlin
niclas.borlin@cs.umu.se

Datavetenskap, Umeå universitet

2023-10-17 Tis

Egendefinierade poster

- Koden

```
typedef struct {
    type1 id1;
    type2 id2;
    ...
} type_name;
```

  definierar en datatyp `type_name` som är en post (*record*, `struct`)
- Posten `type_name` består av delar som kallas fält (eng: *field, member*)
  - `id1` av typ `type1`,
  - `id2` av typ `type2`, osv.
- Alla poster av typen `type_name` har samma delar

Resultathantering

- Antag att vi vill skriva ett program som hanterar studenter och deras resultat, samt klarar av kravet på anonymitet
- Till att börja med behöver vi en datatyp för att hantera resultat anonymt:

```
typedef struct {
    int id;
    double score;
    char grade;
} student_result;
```

- För att utföra de operationer vi behöver definierar vi funktioner som har objekt av typen `student_result` som in- och/eller utdata

Resultatfunktioner

```
student_result create_student_result(int id, double score,
                                     char grade)
{
    student_result sr;
    sr.id = id;
    sr.score = score;
    sr.grade = grade;
    return sr;
}
void print_student_result(student_result sr)
{
    printf("%4d%7.1f%4c", sr.id, sr.score, sr.grade);
}
```

struct-datatyper

- Punktoperatorn (.)
  - Används för att komma åt enskilda värden i en `struct`
  - Ex. `sr.id = id`
- En `struct` kan tilldelas en annan `struct` om de är av samma typ, ex:

```
student_result sr1 = {1, 53, '5'}, sr2;
sr2 = sr1;
```

- En `struct` kan både skickas som parameter till en funktion och returneras från en funktion

Studenter

- När tentan är färdigrättad behövs en datatyp för en student som innehåller ett namn och resultat:

```
#define NAME_LENGTH 100
typedef struct {
    char name[NAME_LENGTH];
    student_result result;
} student;
```

- En struktur `s1` av typen `student` representerar en student vars namn anges av strängen `s1.name` och vars resultat finns i fältet `s1.result`

Studentfunktioner

- Nu kan vi också definiera funktioner som opererar med studenter
- Till exempel:

```
student create_student(char *name, student_result sr)
{
    student s;
    strncpy(s.name, name, NAME_LENGTH);
    s.result = sr;
    return s;
}
void print_student(student s) {
    printf("%-25s", s.name);
    print_student_result(s.result);
}
```


Vi repeterar pekare

- Koden:

```
int a;
int *ap;
ap = &a;
a = 5;
```

får exakt samma resultat som:

```
int a;
int *ap;
ap = &a;
*ap = 5;
```

Pekare till struktur

```c
#include <stdio.h>
typedef struct {
    double x;
    double y;
} point;
void reflectPoint(point *p)
{
    double temp;
    temp = (*p).x;
    (*p).x = (*p).y;
    (*p).y = temp;
}
int main(void)
{
    point p1 = {3.5, 7.34};
    reflectPoint(&p1);
    printf("p1.x = %f, p1.y = %f\n", p1.x, p1.y);
    return 0;
}
```


Linjärsökning

- Algoritm med vanliga ord:
  - Starta från början och sök tills elementet hittats eller sekvensen tar slut

Binärsökning

- Kräver att sekvensen är sorterad
- Algoritm med vanliga ord
  1. Jämför med elementet närmast mitten i sekvensen
     - 1.1 Om likhet — klart
     - 1.2 Om det sökta värdet kommer före elementet närmast mitten, sök i den vänstra delsekvensen, hoppa till steg 1
     - 1.3 Om det sökta värdet kommer efter elementet närmast mitten, sök i den högra delsekvensen, hoppa till steg 1

Insertion sort av fält

- Algoritmen i grova drag:
  - Börja med ett element (ett element är sorterat)
  - Ta sedan ett element i taget och sortera in på rätt plats bland de tidigare sorterade elementen

Bubble Sort

- Algoritmen i grova drag:
  - Upprepa följande tills ingen förändring sker:
    - Jämför alla elementen ett par i taget
      - Börja med element 0 och 1, därefter 1 och 2, osv
    - Om elementen är i fel ordning, byt plats på dem

Merge Sort

- Algoritmen i grova drag
  - Om sekvensen har ett element
    - Returnera sekvensen (den är redan sorterad)
  - annars
    - Dela sekvensen i två ungefär lika stora delsekvenser
    - Sortera delsekvenserna rekursivt
    - Slå samman delsekvenserna (*Merge*)
    - Returnera den sammanslagna sekvensen

Rekursion

- Det finns inget som hindrar att en funktion anropar sig själv
  - Detta kallas rekursion
- För att rekursionen skall terminera (avslutas), måste det
  1. finnas ett eller flera stoppvillkor (basfall) och
  2. varje rekursivt anrop måste ta oss minst ett steg närmare ett stoppvillkor

Ett exempel

- Skriva ut innehållet i en array aha rekursiv funktion
- Flera varianter vi börjar med en som gör samma sak som:

```
void print_array(int n, int arr[]) {
    for (int i = 0 ; i < n ; i++) {
        printf("%d ", arr[i]);
    }
}
```

Strängar i C

- C har ingen egen datatyp för strängar
- I stället representeras strängar som fält av `char`
- En sträng i C är noll-terminerad, dvs. den avslutas (termineras) med tecknet *null* (som har värdet 0)
- Vi har två längder att hålla reda på:
  - En fix (maximal) längd som avgörs vid fältets deklaration
  - En variabel längd som bestäms av första \0-tecknet
- Termineringstecknet \0 måste rymmas inom det allokerade utrymmet

```
char s3[5] = "abc";
```


Två viktiga frågor

- Är målsträngen (-bufferten) tillräckligt stor?
- Slutar den skapade strängen med '\0'?

- Se upp för *buffer overflow*!
- Använd de funktioner där man kan ange max antal!

Funktioner för strängar

- Kopiera sträng
  - `strcpy`, `strncpy`
- Jämföra strängar
  - `strcmp`, `strncmp`
- Kontrollera längd på sträng
  - `strlen`
- Konkatenera (slå ihop strängar)
  - `strcat`, `strncat`

Get strings — `fgets` och `gets`

- Vill vi läsa in strängar kan vi använda funktionen `fgets`

```
char *fgets(char *str, int max_len, FILE *filep);
```

- Det finns också en funktion som heter `gets`

```
char *gets(char *str);
```

- Använd INTE `gets`!
  - `gets` läser från `stdin` (normalt tangentbordet) tills den påträffar '\n' eller `EOF` (*end-of-file*)
  - Risk för buffer overflow
- `fgets` är säkrare!
  - Läser maximalt `max_len-1` tecken
  - Avslutar alltid med '\0', sparar '\n' bara om det ryms

getchar och putchar

- Funktionen `getchar` läser ett tecken från `stdin` (normalt tangentbordet)
- Funktionen `putchar` tar ett tecken och skriver det till `stdout` (normalt skärmen)

Från sträng till tal

- Funktionen `scanf` har vi använt många gånger
  - Vi kan läsa in till en sträng – `'%s'`
  - Vi kan läsa in till olika typer av tal – `'%d'`, `'%lf'`

```
scanf("%d", &n);
```

- Funktionen `sscanf` fungerar som `scanf` men läser från en buffert istället för `stdin`

```
sscanf(buf, "%lf", &x);
```

- Används gärna i kombination med `fgets`:

```
char buf[BUFSIZE];
double x;
fgets(buf, BUFSIZE-1, stdin);
sscanf(buf, "%lf", &x);
```

- Fördelen är att om något går fel i `sscanf` vid tolkningen av `buf` så ligger inte "skräpet" kvar vid nästa anrop till `fgets`

Från tal till sträng

- Funktionen `printf` har vi också använt många gånger
  - Vi kan skriva ut olika värden
  - Vi kan skriva ut en sträng – `'%s'`

```
printf("%s %s\n", "Hello", name);
```

- Funktionen `sprintf` fungerar som `printf` men skriver till en buffert istället för till `stdout`

```
sprintf(str, "The answer is %f", x);
```


Tolka tecken

- Ska vi tolka indata kan det vara bra att kunna klassificera enskilda tecken
  - Header-filen `ctype.h` innehåller funktioner för att kolla på tecken
  - Följande funktioner returnerar 0 vid falskt
    - `isalpha(c)` kollar om c är en bokstav
    - `isdigit(c)` kollar om c det är en siffra
    - `islower(c)` kollar om c är en liten bokstav
    - `isupper(c)` kollar om c är en stor bokstav
    - `isspace(c)` kollar om c är ett blanksteg (eller tab, newline, m.fl.)
    - `ispunct(c)` kollar om c är ett tecken som inte är ett kontrolltecken, ett blanksteg, en bokstav eller en siffra

Manipulera tecken

- Header-filer `ctype.h` innehåller även funktioner för att manipulera (engelska) tecken
  - `tolower(c)` ändrar `c` till liten bokstav
  - `toupper(c)` ändrar `c` till stor bokstav

Matlab

- Matlab-delen startar nästa måndag