FACIT!!

Omtentamen - Programmering

DVA117

Akademin för innovation, design och teknik

Onsdag 2014-01-15

Skrivtid: 14.10 – 17.30
Hjälpmedel: Inga
Lärare: Caroline Uppsäll, 021-101456
Robert Suurna, 021-151790

Preliminära betygsgränser
Betyg 3: ??p
Betyg 4: ??p
Betyg 5: ??p
Max: ??p

Bonuspoäng
Inga bonuspoäng kan tas med till omtentamen.

Allmänt

- All kod skall skrivas i standard ANSI C.
- Påbörja varje ny uppgift på nytt blad och skriv bara på ena sidan av pappret. Deluppgifter (a, b, c,…) kan skrivas på samma blad
- Referera inte mellan olika svar.
- Om du är osäker på vad som avses i någon fråga, skriv då vad du gör för antagande.
- *Oläsliga/oförståeliga/ostrukturerade svar rättas inte.*
- Kommentera din kod!
- Tips: Läs igenom hela tentan innan du börjar skriva för att veta hur du ska disponera din tid.

Lycka till!

FACIT!!

Prioritet och associativitet hos operatorerna i C

Högsta prioritet högst upp i tabellen, lägsta prioritet längst ner i tabellen. Grupperade operatorer har samma prioritet.

Table A.5 Summary of C Operators (Programming in C, Stephen G. kochan)

Operator	Description	Associativity
`()`	Function call	
`[]`	Array element reference	
`->`	Pointer to structure member reference	Left to right
`.`	Structure member reference	
`-`	Unary minus	
`+`	Unary plus	
`++`	Increment	
`--`	Decrement	
`!`	Logical negation	
`~`	Ones complement	Right to left
`*`	Pointer reference (indirection)	
`&`	Address	
`sizeof`	Size of an object	
`(type)`	Type cast (conversion)	
`*`	Multiplication	
`/`	Division	Left to right
`%`	Modulus	
`+`	Addition	Left to right
`-`	Subtraction	
`<<`	Left shift	Left to right
`>>`	Right shift	
`<`	Less than	
`<=`	Less than or equal to	Left to right
`>`	Greater than	
`=>`	Greater than or equal to	
`==`	Equality	Left to right
`!=`	Inequality	
`&`	Bitwise AND	Left to right
`^`	Bitwise XOR	Left to right
`|`	Bitwise OR	Left to right
`&&`	Logical AND	Left to right
`||`	Logical OR	Left to right
`?:`	Conditional	Right to left
`= *= /= %= += -= &= ^= |= <<= >>=`	Assignment operators	Right to left
`,`	Comma operator	Right to left

Uppgift 1 [7p]

a) Vad har preprocessorn för uppgift när ett C-program kompileras?
(1p): Preprocessorn är det första steget i kompileringen av ett C-program. Den tar hand om alla rader som inleds med tecknet #.
(Ett separat program som översätter en källfil till en ny källfil där alla preprocessorinstruktioner har exekverats. Preprocessorinstruktioner är speciella instruktioner i C-källkoden som används för att styra kompileringen (villkorlig kompilering), definiera konstanter och makron samt inkludera huvudfiler.)

b) Varför deklarerar vi variabler i C, dvs ger dem en viss typ (ange minst två anledningar)?
(2p): För att hålla reda på hur vi ska tolka det som finns i minnet (ettorna och nollorna). För att begränsa minnesåtgången. För att få programmen att exekvera snabbare.

c) Beskriv kortfattat begreppet *makro* och hur det används i C-kod. Ta även med ett exempel på hur man skriver ett makro.
(2p): Ett makro är en identifierare som definierats att representera en viss textsträng. När preprocessorn träffar på ett makro i koden byts det ut mot textsträngen (kallas för makroexpansion). `#define PI 3.14`

d) Minnesmodellen för ett C-program innehåller en del som vi kallar för *heap.* Vad används den till?
(1p): Heapen lagrar dynamisk data, programmeraren sköter själv minneshanteringen.

e) Hur fungerar variabeler som är deklarerad med lagringsklassen `static`?
(1p): Den får ett minnesutrymme som existerar under hela programmets exekvering. Den kommer att minnas sitt värde mellan funktionsanrop.

Uppgift 2[4p]

Vilka värden har följande uttryck i C (vad evakuerar varje uttryck till)?
För samtliga uttryck gäller: `int x=40, y=2, z=4, a=1, b=6;`

a) `z > 8 || x == y * 20`

(1p): `0 || 40 == 2*20  0 || 1  1`

b) `(a != x || b != y) && !(b < x && a == 1)`

(1p): `(1 || 1) && !(1 && 1)  1 && 0  0`

c) `b >= a * 5 && y == ((x / 4) - 8) && !a`

(1p): `1 && 1 && !1  1 && 1 && 0  0`

d) `x - a > b || z + a < b + x || z <= y`

(1p): `39 > 6 || 5 < 46 || 4 <= 2  1 || 1 || 0  1`

Uppgift 3 [3p]

Antag följande rader kod:

```
char str[38]="Kan jag";
strcat(str," programmera C?\n");
```

a) Vilket specialtecken bör alla strängar avslutas med för att fungera ihop med *str*-funktioner?
(0.5p): \0

b) Vilken längd har strängen, dvs. vad returnerar `strlen(str)`?
(0.5p): 23 (\n räknas med)

c) Vad skrivs ut av satsen: `printf("%c",str[8]);`
(0.5p): p

d) Hur kan man ”tömma” (reset) strängen `str`?
(0.5p): `str[0]='\0'`, eller `*str='\0'`, eller `memset(str, 0, sizeof(str));` eller med en for-loop.

e) Hur många bitar (bits) används i minnet för variabeln `str` när båda raderna körts?
(1p): 304 bits (38 byte * 8 bits/byte)

Uppgift 4 [10p]

Er lärare behöver ett program som kan hålla ordning på studenternas resultat i en kurs. För varje student i kursen ska man kunna spara namn (förnamn och efternamn), personnummer samt betyg på momenten LAB (U eller G) och TEN (U, 3, 4 eller 5) samt slutbetyget i kursen (3, 4 eller 5).

a) Skriv koden för en lämplig typ (struct) som kan hålla informationen om en student i klassen enligt beskrivningen ovan. (2p)
b) Skriv koden för att skapa ett fält (array) för kursen DVA117 som kan hålla 65 studenter genom att använda funktionen `calloc()`. Felhantering för allokering av dynamiskt minne är ett krav. (2p)
c) Skapa en funktion som sätter betyget på momentet TEN. Funktionen ska som inparametrar minst ta studentens personnummer samt betyget. Returvärde från funktionen ska vara 1 om operationen lyckades och 0 om den inte lyckades. Ni kan här anta att arrayen från b) är fylld med data. (4p)
d) Skriv en kodrad som tilldelar betyget ’G’ på momentet LAB till studenten på den sista posten. Använd piloperatorn. (2p)

Lösningar

a) (2p)

```
typedef struct{
   char fornamn[20];
   char efternamn[20];
   long personnummer;
   char betygLAB;
   char betygTEN;
   char slutbetyg;
   }student;


   //alternativt utan typedef, då ska detta följa i nedanstående
   uppgifter också
```

b) (2p)

```
student *dva117 = (student *)calloc(65, sizeof(student));
if(dva117 == NULL)
{
    printf("kunde inte allokera minne");
    return 0;
}
```

c) (4p)

```
int sattBetygTEN(long pnr, char betyg, student *arr); (returväde
1p, parameterlista 1p)
{
    int i;
```

```
    for(i = 0; i < strlen(arr); i++) (1p)
    {
        if(arr[i].personnummer == pnr) (1p)
        {
            arr[i].betygTEN = betyg;
            return 1;
        }
    }
    return 0;
}
```

d) (2p)

```
(dva117 + 64)- > betygLAB  =  'G';
```

Uppgift 5 [3p]

Skriv om switch-satsen i koden nedan till en *if-else-if*-sats.

```
int main(void)
{
    int val = 0;

    puts("MENY");
    puts("------------------------------");
    puts("1 - Lägg till post");
    puts("2 - Ta bort post");
    puts("3 - Sök efter post");
    puts("4 - Skriv ut samtliga poster");
    puts("5 - Avsluta programmet");


    scanf("%d", &val);

    switch(val){
      case 1: laggTill();
              break;
      case 2: taBort();
              break;
      case 3: sok();
              break;
      case 4: skrivUt();
              break;
      case 5: avsluta();
              break;
      default: puts("Felaktigt val");
    }

    return 0;
}
```

Skriv om denna del

(1p) för logiska uttrycken | (1p) för else if | (1p) för else

```
if(val == 1){
    laggTill();
}
else if(val == 2){
    taBort();
}
else if(val == 3){
    sok();
}
else if(val == 4){
    skrivUt();
}
else if(val == 5){
    avsluta();
}
else {
    puts("Felaktigt val");
}
```

Uppgift 6 [10p]

Delfråga a till b nedan handlar om följande programkod.

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

int main(void)
{
    float summa = 0, tal = 0;
    int antal = 0, i = 0, avsluta = 0;
    FILE fp = NULL;
    char filNamn[100] = {};

    printf("Ange namn på fil att läsa: ");
    scanf("%s", filNamn);

    if((fp = fopen(filNamn, "r"))!= NULL)
    {
        printf("Ange antal tal: ");
        scanf("%d", antal);
        i = antal;
        while(i > 0)
        {
            if(fscanf(fp, "%f", &tal)==EOF)
                summa += tal;
            i++;
        }
        printf("Summan: %.4f", summa);
    }
    else
    {
        printf("Filen %s gick inte att öppna, programmet avslutas", filNamn);
        avsluta = 1;
    }

    if(avsluta != 1)
    {
        printf("\nAnge fil att skriva resultat till: ");
        scanf("%s", filNamn);

        if((fp = fopen(filNamn, "w")) != NULL)
            printf(fp, "Summan av de %d första talen är %.f", antal, summa);
        else
        {
            printf("Filen %s gick inte att öppna, programmet avslutas", filNamn);
            avsluta = 1;
        }
        fclose(fp);
    }

    printf("Programmet avslutas");
    return 0;
}
```

a) I koden finns ett antal fel (tankefel och fel som av en kompilator ger error). Det kan också vara någonting som saknas. Hitta felen, skriv ner hela den felaktiga raden och visa hur den ska rättas till (ändras).
Påståenden om fel som i själva verket är korrekt kod ger poängavdrag (lägsta poäng är 0).
(0.5p): `FILE fp = NULL` `//ska vara *fp`
(0.5p): `scanf(”%d”, antal)` `//ska vara &antal`
(0.5p): `if(fscanf(…) == EOF )` `//ska vara != EOF`
(0.5p): `i++` `//ska vara i--`
(0.5p): `//fclose(fp) saknas för första filen som öppnas (r)`
(0.5p): `printf(fp, ”…”);` `//ska vara fprintf(…)`

b) Beskriv vad programmet utför. dvs. vad matas in och vad blir resultatet (inte rad för rad).
(2p): En fil som användaren anger öppnas för läsning. Användaren får ange ett antal och de

första ”antal” talen i filen summeras (om den gick att öppna). Sedan skrivs summan till en fil som användaren väljer (om den filen gick att skapa/öppna).

c) Ge ett exempel från koden på en variabeldeklaration om det finns.
(1p): `float summa` Dvs. någon av variabeldeklarationerna från koden. Vi pratar inte om funktionsdeklarationer som är något annat.

d) Ge ett exempel från koden på en variabelinitiering om det finns
(1p): `t.ex. tal = 0;`

e) Ge ett exempel från koden på en definition om det finns.
(1p): Finns ingen i koden

f) Vad är skillnaden på en textfil och en binärfil? När vill man använda binärfil och när vill man använda textfil?
(2p): En binärfil består av ettor och nollor. Den är alltså mindre krävande för datorn att läsa men ganska svår för oss människor. En textfil består av helt vanliga tecken och har rader. Denna är lättare för oss att läsa/bearbeta men omvandling till binärkod måste ske i datorn.

Använd textfil om användaren behöver bearbeta filen utanför programmet och om det inte gör någonting att programmet blir lite mer krävande. Ska filen endast läsas/skrivas av datorn så använd binärfiler.

Uppgift 7 [4p]

Skriv ett C-program som skriver ut multiplikationstabeller för tabellerna 1 till 5. Varje tabell ska ha intervall 0-10. I din lösning ska du använda *nästlade loopar*. Den ena loppen ska vara av typen *for-loop* och den andra av typen *do-while-loop.*

Mellan varje tabell ska det skrivas ut en linje, men det ska inte skrivas ut en linje högst upp och inte en längst ner.

Skriv ut på formen :

```
1*0= 0
1*1= 1
1*2= 2
1*3= 3
1*4= 0
1*5= 1
1*6= 2
1*7= 3
1*8= 3
1*9= 3
1*10= 3
--------------

2*0= 0
2*1= 2
2*3= 4
```

Osv…

(4p)

```
int main(void){
     int i=1,j;

    do{                                                 //loop1 (1p)
       for(j=0; j<=10; j++)                             //loop2 (1p)
              printf("%d*%d= %d\n",i,j,i*j);            //utskrift (1p)

              if(i!=5)                                  //linje (1p)
                printf("-------------\n");
              i++;
    }while(i<=5);

     return;
}

int main(void){
     int i=1,j;

    for(i=1; i<=5; i++)                                 //loop2 (1p)
    {
        j=0;
        do{                                             //loop1 (1p)
            printf("%d*%d= %d\n",i,j,i*j);              //utskrift (1p)
            j++;
        }while(j<=10);

       if(i!=5)                                         //linje (1p)
            printf("-------------\n");
    }
     return;
}
```

Uppgift 8 [5p]

Nedan finner du ett huvudprogram. Din uppgift är att skriva en funktion `maxfunc`. Ett anrop till funktionen ska göras i huvudprogrammet på markerad plats. Du ska alltså skriva själva funktionen samt anropet. Funktionen måste fungera med resterande huvudprogram. Ändringar till huvudprogrammet får inte göras.

Funktionen

Skriv både metoden och anropet.

```
#define SIZE 5

int main(void)
{
    int arr[SIZE];
    int stor = 0, i;
    float medel = 0;

    for(i = 0; i < SIZE; i++)
    {
        printf("Ange tal: ");
        scanf("%d", &arr[i]);
    }
    //METODANROP TILL maxfunc
    printf("Största talet: %d\nMedeltal: %.2f", stor, medel);
}
```

Lösning (5p):

```
void maxfunc(int arr[], int *stor, float *medel) (2p)
{
    int i, sum = 0;
    *stor = arr[0];
    for(i = 0; i < SIZE; i++)
    {
        sum += arr[i];
        if(arr[i] > *stor) (1p)
            *stor = arr[i];
    }
    *medel = (float)sum/SIZE; (1p)
}
```

Anrop:

```
maxfunc(arr, &stor, &medel); (1p)
```

det går också bra att returnera den ena av stor och medel och skicka den andra som pekare. Anropet måste självklart stämma med den skrivna funktionen.