Programmering, grundkurs, 8.0 hp HI1024, omtentamen, TEN1

Tisdagen den 7 juni 2011, 8.15 – 12.15

Tentamen består av två delar, del A och del B. Del A innehåller 10 kryssfrågor på olika teman inom *C*-programmering. Varje fråga är uppbyggd på ett påstående. Påståendet är antingen *helt korrekt* och är då alltså SANT, eller så finns *minst ett allvarligt fel* i påståendet och då är alltså påståendet FALSKT. Man kryssar det man tror på och om man kryssar rätt får man 1 poäng. Kryssar man fel får man –1 poäng. Det är alltså riskabelt att chansa. Man har alltid alternativet att kryssa AVSTÅR som då ger 0 poäng. Del A kan inte ge mindre än 0 poäng totalt. (Om man har mer fel är rätt får man alltså inga negativa poäng.) Del B innehåller 10 traditionella frågor som ni ska svara skriftligt på. Totalt antal poäng är 37. Gränsen för E är 16 och gränsen för Fx är 14. KTHs allmänna regler för examination gäller. (Gränserna är ungefärliga.)

Del A

Påstående 1. Ett *C*-program ska alltid kompileras och länkas innan det kan köras. Vid kompilering kan så kallade *kompileringsfel* förekomma. Det betyder att ett program inte skrivits på *helt* rätt sätt. Det här behöver inte vara ett problem för om man delat upp sitt *C*-program i väl uttänkta funktioner och ett kompileringsfel förekommer i en funktion som aldrig anropas så kan programmet exekveras ändå, just eftersom den funktion där kompileringsfelet förekom aldrig anropas. Detta illustrerar varför det är bra att använda funktioner då man programmerar.

[] SANT [] FALSKT [] AVSTÅR

Påstående 2. Man kan skicka arrayer av heltal till funktioner som parametrar genom funktionsprototypen

```
funk (int arr[], int length).
```

Längden på arrayen måste då skickas separat i parametern `length`. Man kan uppfatta strängar som arrayer av tecken där sista tecknet är NULL-tecknet. Det innebär att funktioner som opererar på strängar (som är korrekt formade) inte behöver längden skickade till sig som funktionen `funk` ovan.

[] SANT [] FALSKT [] AVSTÅR

Påstående 3. Om variabeln `pris` är deklarerad som ett heltal och en användare i något skede av ett program ska mata in ett värde i denna variabel kan det göras med hjälp av funktionen `scanf`. Denna funktion tar en formatsträng och adressen till `pris` som parametrar, formatsträngen kan innehålla `%d` för inmatning av heltal. Men om man skriver fel, till exempel `scanf("%f", &pris);` anger detta att man istället ska mata in ett flyttal i heltalsvariabeln `pris`. Det är dock ingen fara för vid programkörningen läser visserligen `scanf` in ett flyttal från användaren, men det sker en automatisk omvandling till heltal så att rätt värde kommer in i variabeln `pris`.

[] SANT [] FALSKT [] AVSTÅR

Påstående 4. Tecken i *C*, alltså de som deklareras som `char`, kan hanteras som små heltal. För de små engelska bokstävernas del, alltså tecknena från `'a'` till `'z'` som har asciikoderna 97 till och med 122, betyder det att vi kan hantera dessa bokstäver som talen 97 till 122. För siffrornas del alltså tecknena, `'0'` till och med `'9'`, gäller att de har har asciikoderna 48 till och med 57 och siffrorna kan alltså behandlas som dessa tal. Till exempel kan vi läsa in ett tecken med `scanf`, så

här: `scanf("%c", &tecken);`, och om det tecknet är en siffra (alltså någon av `'0'` till `'9'`) så kan vi omvandla det till ett motsvarande heltal genom att subtrahera 48, vi kan då uttrycka 48 som tecknet `'0'` och skriva

```
siffer_varde = tecken - '0';
```

på samma sätt, om `tecken` är en liten bokstav kan vi omvandla det till en stor bokstav genom att subtrahera 32, alla stora bokstäver har nämligen asciikoder som ligger 32 steg under asciikoderna för de stora bokstäverna. *(Ledning: En ascii-tabell finns längst bak i tentan. Den är korrekt.)*

[] SANT [] FALSKT [] AVSTÅR

Påstående 5. Antag att vi har deklarationen

```
struct point {
  double x, y, z;
};
```

Denna struct kan då användas för att representera en punkt i ett tredimensionellt rum. Vi skulle då skriva

```
struct point p;
struct point *p1;
p.x = 10; p.y = 20; p.z = 30;
p1 = &p;
```

Detta innebär att vi skapar en pekare `p1` som pekar på variabeln `p`. Vi kan då komma åt medlemmarna i `p` genom att skriva `(*p1).x`, `(*p1).y` och `(*p1).z`, men det blir lite knöligt. Därför har man infört pilnotationen: `p1->x`, `p1->y` och `p1->z` är synonyma med `(*p1).x`, `(*p1).y` och `(*p1).z`.

[] SANT [] FALSKT [] AVSTÅR

Påstående 6. Operatorn `&` kallas adressoperatorn och `*` kallas avreferensieringsoperatorn. De är på sätt och vis varandras motsatser, om `x` är en heltalvariabel så är `&x` adressen till den variabeln och uttrycket `*(&x)` ska då utläsas "ta fram det värde som finns på platsen som indikeras av `&x`" vilket är variabeln `x` innehåll. Alltså, satserna `x=0; printf("%d", *(&x));` ger utskriften 0 eftersom `x` har det värdet. Det här fungerar åt andra hållet också, vi skulle kunna skriva `x=0; printf("%d", &(*x));` och återigen få utskriften 0. *(Ledning: Att `*` kallas avreferensieringsoperatorn (eng. dereferencing operator) är sant, vi har inte talat så mycket om vad den heter i kursen.)*

[] SANT [] FALSKT [] AVSTÅR

Påstående 7. Låt `well_formed` vara en testfunktion som opererar på en viss datatyp som returnerar sant eller falskt (här tolkat som 1 eller 0). Studera då dessa två villkor

```
!(well_formed(&a) || well_formed(&b) || well_formed(&c))

(!well_formed(&a) && !well_formed(&b) && !well_formed(&c))
```

Dessa två villkor är exakt likvärdiga, det vill säga vi kan använda vilken vi vill av dem där villkor behövs, som till exempel i `if`-satser eller loopar.

[] SANT [] FALSKT [] AVSTÅR

Påstående 8. Om man kör `if`-satsen

```
if(a=1)
  printf("a = 1");
```

Så kommer alltid utskriften `a = 1` oberoende av vilket värde `a` hade från början.

[] SANT [] FALSKT [] AVSTÅR

Påstående 9. Studera nedanstående program

```
#include <stdio.h>

int multiply (int x, int y) { return x * y; }

int main()
{
  int x;
  int y;

  printf( "Input two numbers to be multiplied: " );
  scanf( "%d", &x );
  scanf( "%d", &y );
  printf( "The product of your two numbers is %d\n", multiply( x, y ) );
  getchar();
}
```

Programmet hanterar heltal (`int`) och om man vill att programmet ska kunna hantera flyttal behöver man bara ändra i funktionens deklaration, till `(float x, float y)`, man kan använda heltalsvariabler för att lagra flyttal, det finns automatiska inbyggda omvandlingfunktioner i *C* för just detta. Programmet kommer därför att fungera med flyttal i anropet till funktionen, man behöver alltså inte ändra på deklarationen av `x` och `y` i huvudprogrammet.

[] SANT [] FALSKT [] AVSTÅR

Påstående 10. Studera nedanstående `switch`-sats:

```
switch(day)
{
  case MONDAY: case TUESDAY: case WEDNESDAY: case THURSDAY: return 100.0;
  case FRIDAY: return 150.0;
  case WEEKEND: return 200.0;
}
```

Den kan också skrivas

```
switch(day)
{
  case FRIDAY: return 150.0;
  case WEEKEND: return 200.0;
  default: return 100.0;
}
```

Denna sats fungerar på precis samma sätt som den förra, förutsatt att `day` är en heltalsvariabel som bara antar värdena `MONDAY`, `TUESDAY`, `WEDNESDAY`, `THURSDAY`, `FRIDAY` och `WEEKEND`. För frågans sammanhang antas dessa vara skilda heltalskonstanter, till exempel med värdena 0, 1, 2, 3, 4, 5, 6.

[] SANT [] FALSKT [] AVSTÅR

Del B. (Nödvändiga `#include`-direktiv är utelämnade, men ska anses finnas där ändå.)

Uppgift 11. (2p) Studera nedanstående program:

```
main()
{
  char ch[] = {'a','b','c','d','e','f','g','h','i','j','k','l'};
  int i;
  for(i=1;i<=5;i++)
    printf("%c", ch[(i*13)%11]);
  printf("\n\n");
}
```

Ange den utskrift som uppkommer då man kör det.

Uppgift 12. (4p) Studera nedanstående program:

```
main()
{
  char ch[] = {'a','b','c','d','e','f','g','h','i','j','k','l'};
  int i, j, m; char tmp;

  ch[11]=0;

  for(i=0;i<3;i++)
  {
    m=i;
    for(j=i+1;j<11;j++)if(ch[m]<ch[j])m=j;
    if(m!=i)
    {
      tmp=ch[i];
      ch[i]=ch[m];
      ch[m]=tmp;
    }
  }

  printf("%s.\n",ch);
}
```

Ange den utskrift som uppkommer då man kör det. (Ledning: Var noggrann med att ta hänsyn till att det står en **3**:a i koden ovan.)

Uppgift 13. (2p) Studera nedanstående program:

```
main()
{
  int i;
  for(i=1;i<=...;i++)
    printf("%d ", ...);
  printf("\n");
}
```

Programmet är ofullständigt. Man vill att programmet ska skriva ut de tal som bildas genom att

subtrahera de fyra första kvadraterna från de fyra första kuberna. Kvadrater är andra potensen av ett tal, de fyra första kvadraterna är alltså 1 upphöjt till 2 (som är 1), 2 upphöjt till 2 (som är 4), 3 upphöjt till 2 (som är 9) och 4 upphöjt till 2 (som är 16). Kuber är tredje potensen av tal, och de fyra första kuberna är 1, 8, 27 och 64. Fyll i vad det ska stå där det står "`...`" så att programmet fungerar enligt beskrivningen ovan. Ange också vad programmet ger för utskrift.

Uppgift 14. (2p) Studera nedanstående program

```
main()
{
  int i, b[] = { 5, 4, 3, 0, 2, 1 };
  char tecken[] = "BAR KO";

  for(i=0;i<6;i++)
    printf("%c",tecken[b[i]]);

  printf("\n");

}
```

Ange programmets utskrift.

Uppgift 15. (3p) Studera nedanstående program:

```
int f1(int a, int *b)
{
  a=a+5;
  (*b)++;
  printf("F1 %d %d \n", a, *b);
  return a;
}

f2(int *y, int x)
{
  int z;
  (*y)++; x--;
  z=f1(*y, &x);
  printf("F2 %d %d %d\n", x, *y, z);
}

main()
{
  int x=1,y=2,z=3;
  f2(&x,y+z);
  printf("MAIN %d %d\n", x, y+z);
}
```

Ange den utskrift som uppkommer då man kör det.

Uppgift 16. (3p) Tre heltal, a, b och c kallas en *pythagoreisk trippel* om de uppfyller a*a + b*b = c*c, detta innebär att de kan uppfattas som längderna av tre sidor i en rätvinklig triangel, a och b blir kateterna och c blir hypotenusan. Nedanstående program är tänkt att utröna om tre inmatade tal ugör en pythagoreisk trippel eller ej. Men programmet innehåller tre fel. Finn dessa fel och rätta dem. Felen kan sägas finnas på tre olika rader. Det kan finnas fler än ett fel på en rad.

```
int pyth_triple ( int a, int b, int c)
{
  return (c*c == a*a + b*b)
}

main() {
  int a, b, c;
  printf("Mata in a, b och c: "); scanf("%d %d %d", &a, &b, &c);

  if(pyth_trippel(&a,&b,&c))
    printf("Pythagoreisk trippel.\n");
  else
    printf("Inte pythagoreisk trippel.\n");

}
```

Uppgift 17. (2p.) I *Del A* förekom flera falska påståenden. Välj två av dessa falska påståenden och förklara vad som egentligen gäller i sammanhanget, det vill säga uttryck vad som skulle vara sant i sammanhanget och varför det är på det viset.

Uppgift 18. (3p) Betrakta nedanstående program, det slumpar fram fem strängar och vill sortera dem i bokstavsordning:

```
random_str(char str[]) {
  int pos;
  for(pos=0;pos<9;pos++)
    str[pos]='a'+rand()%25;
  str[pos]='\0';
}

main(){
  char str[5][10];
  char tmp[10];
  int i, j, min=0;
  srand(time(0));
  for(i=0;i<5;i++)random_str(str[i]);
  for(i=0;i<5;i++)printf("test: %s\n", str[i]);

  //Sortering:

  for(i=0;i<5;i++) {
    min=i;
    for(j=i+1;j<5;j++)
      if(strcmp(str[min],str[j])>0)j=min;
    if(min!=i)
    {
      strcpy(tmp,str[i]);
      strcpy(str[i],str[min]);
      strcpy(str[min],tmp);
    }
  }

  printf("\n");
  for(i=0;i<5;i++)printf("test: %s\n", str[i]);
}
```

Det finns dock ett fel någonstans i sorteringen, finn felet och rätta det! Det är inte ett kompileringsfel, det går att köra programmet, men programmet hamnar i en oändlig loop.

Uppgift 19. (2p) I *Del A* förekom flera falska påståenden. Välj ytterligare två av dessa falska påståenden och förklara vad som egentligen gäller i sammanhanget, det vill säga uttryck vad som skulle vara sant i sammanhanget och varför det är på det viset. (Du måste förstås välja andra påståenden än du redan behandlat i uppgift 17.)

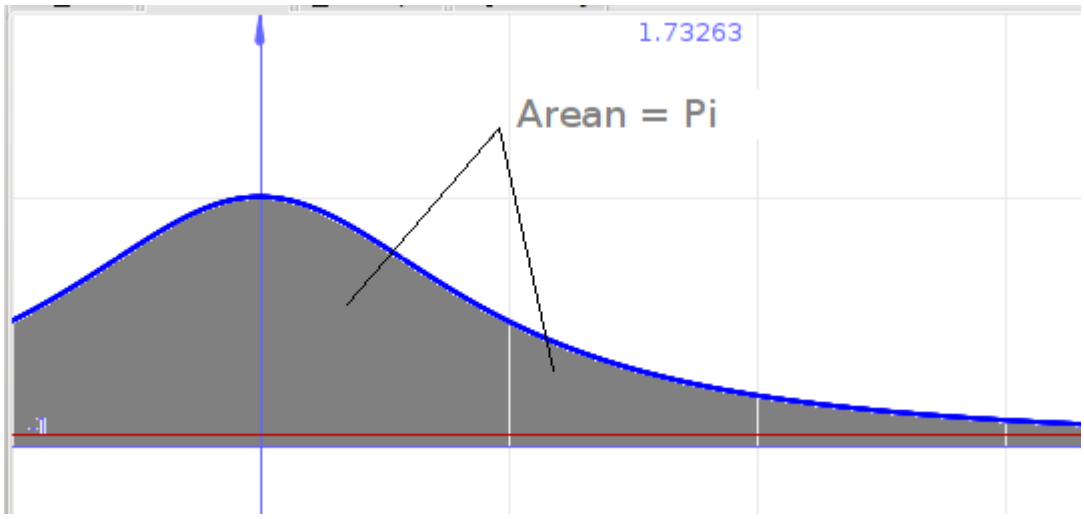
Uppgift 20. (4p) Studera nedanstående program:

```
double f ( double x ) {
  return 1 / ( 1 + x*x );
}

main() {
  int i, delar = 1;
  double integral_varde = 0.0, dx;

  while(delar<256)
  {
    integral_varde = 0.0;
    dx = 1.0 / delar;
    for(i=0;i<delar;i++)
    {
      integral_varde += f(dx*(i+1))*dx;
    }
    printf("Delar:\t%d\tI:\t%f\t%f\t%f.\n",
               delar,integral_varde,2*integral_varde,4*integral_varde);
    delar = delar * 2;
  }
}
```

Här är dess utskrift: *(tabulatorstegen fungerar inte riktigt, men bortse från det.)*

```
Delar:  1       I:      0.500000        1.000000        2.000000.
Delar:  2       I:      0.650000        1.300000        2.600000.
Delar:  4       I:      0.720294        1.440588        2.881176.
Delar:  8       I:      0.753497        1.506994        3.013988.
Delar:  16      I:      0.769610        1.539221        3.078442.
Delar:  32      I:      0.777545        1.555090        3.110180.
Delar:  64      I:      0.781482        1.562963        3.125927.
Delar:  128     I:      0.783442        1.566885        3.133770.
```

Syftet med programmet är att beräkna ett närmevärde på Pi som är 3.1415926 etc. Vi ser att vi kommer nära Pi i sista kolumnen, det står 3.13 och detta är ett värde som kommer att växa ju längre vi låter programmet köra. Tanken bakom programmet är att beräkna arean under grafen till hörande till funktionen 1/(1+x*x). Som ni kanske minns från gymnasiet är arean under denna graf lika med just Pi. Det betyder att om vi väljer ett tillräckligt stort intervall av alla reella tal och beräknar arean under grafen så kommer vi att komma nära Pi. I vårt program ovan väljer vi att beräkna arean under grafen för positiva värden på x och sedan multiplicerar vi resultatet med 2. Det är areaberäkningarna multiplicerade med 2 som vi ser i mittenkolumnen, det är de värden som svarar mot utskriften "`2*integral_varde`" i `printf`-satsen i programmet. MEN: vi närmar oss inte Pi i denna följd av värden! Vi närmar oss halva Pi! Varför det? Det finns ett problem någonstans i programmet som är givet, det fungerar inte riktigt som det är tänkt. Finn felet och förklara hur programmet ska rättas till så att vi verkligen närmar oss Pi i mittenkolumnen. Den tredje kolumnen är att betrakta som en testutskrift, vi ser att vi närmar oss Pi här, vi undrar varför vi inte får Pi i mittenkolumnen. På nästa sida ser vi en graf som illustrerar den area vi vill använda oss av i denna uppgift.



Ascii-tabellen

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com