

**GU / Chalmers Campus Lindholmen Tentamen Programutveckling
2016-01-13 LEU 482 / TIG167**

Examinator:	Henrik Sandklef (0700-909363)
Tid för tentamen:	2016-01-13, 08.30 – 12.30
Ansvarig lärare:	Henrik Sandklef, besöker salen ca 09.30 & 11.30
Hjälpmedel:	Inga hjälpmedel tillåtna utöver bilagor i tentamenstesen.
Betygsgränser CTH:	Max 50 poäng / 3: 20–30, 4: 31–40, 5: 41–50
Betygsgränser GU:	Max 50 poäng / G: 25, VG: 37.5

Allmänna instruktioner:

- använd helst engelska namn på dina variabler etc
- om du inte förstår en fråga, gå vidare till nästa och fråga läraren när denna/denne besöker
- skriv tydligt
- motivera väl
- om du inte kommer ihåg vad en viss c-funktion heter, vilka parametrar den tar o s v : skriv det du kommer i håg och kommentera
- max en uppgift per blad
- glöm inte kolla eventuell dålig indata i dina funktioner
- om du ombeds att bara skriva en funktion behöver du inte skriva ett program (med main-funktion)

1 (Build tools etc)

Förklara vad följande begrepp innebär (1p). Det räcker med ca 2-3 meningar som svar på respektive deluppgift:

a) kompilering eller kompilator

b) program

Totalt: 1p

LÖSNINGSFÖRSLAG

Kompilator är det program som gör om källkod (t ex skriven i C) till något körbart (t ex ett program). Men delar ibland in en C-kompilators jobb i pre-processing, kompilering och länkning. Det finns andra sätt att dela in en kompilator.

Program är en passiv serie (sekventiella) instruktioner som styr en dator. När man kör ett program skapas en process.

2 (Types, variables, assignment, operators)

a) Om du vill lagra en persons ålder i en variabel, vad vore en bra typ för detta. Deklarera en variabel med lämpligt namn av denna typ. (1p)

b) Tilldela variabeln ovan ett värde som anger att åldern är 40. (1p)

c) Vad är (de respektive) värdena av följande uttryck (2p):

$$4 + 3$$

$$4 - 3$$

$$4 / 3$$

$$4 \% 3$$

Totalt: 4p

LÖSNINGSFÖRSLAG

a) int

I vardagen pratar man sällan om en persons ålder som 30.45 år så ett heltal räcker gott.

```
int age;
```

b)

```
int age = 40;
```

c)

$$4 + 3 \Rightarrow 7$$

$$4 - 3 \Rightarrow 1$$

$$4 / 3 \Rightarrow 1$$

4 % 3 => 1

3 (Expressions and typecast)

a) Vad är värdet på variabeln `nr_of_apes` efter följande kod har körts (1p):

```
int nr_of_chimps = 13 ;
int nr_of_gorillaz = 17 ;
int nr_of_apes = nr_of_chimps + nr_of_gorillaz;
nr_of_apes++;
nr_of_apes++;
```

b) Kolla på följande kod

```
int a;
int b;
char res;
.... some more code assigning
.... and using the ints above ...
res = a+b;
```

Koden ovan har (åtminstone) ett fel. Beskriv det, skriv en lösning och motivera ditt svar (vilket du i och för sig alltid skall göra). (3p)

Totalt: 4p

LÖSNINGSFÖRSLAG

a)

```
int nr_of_apes = nr_of_chimps + nr_of_gorillaz;
// nu har nr_of_apes värdet 30
```

```
nr_of_apes++;
// nu har nr_of_apes värdet 31
```

```
nr_of_apes++;
// nu har nr_of_apes värdet 32
```

b)

Du kan inte på ett säkert sätt lagra ett int-värde i en char-variabel. Lösningens förslag:

1. kolla om summan är större än vad som kan lagras i en char. Om det är så varna användaren. Om det är ok kan vi type-casta (typomvandla). `res = (int) (a+b);`
2. Gör om res till en int.

4 (block, function)

```
int nr_of_groups(int students, int students_per_groups) {  
  
    int full_groups = students / students_per_groups;  
    int remaining = students % students_per_groups;  
    if (remaining < 2) {  
        return full_groups;  
    }  
  
    return full_groups+1;  
}
```

Funktionen ovan används för att räkna ut hur många grupper, givet antal studenter och önskat antal studenter i en grupp, som finns i en klass. Funktionen skulle kunna användas för att automatiskt skapa grupper i ett web-system (tänk PingPong/GUL).

- a) Förklara hur funktionen fungerar (1p).
- b) Vad händer i program när man skriver return? (1p)
- c) Vad blir returnerade värdet om man ger funktionen värdena 7 respektive 4 (1p)?
- d) Skriv kod som använder funktionen och sparar undan det uträknade antalet grupper i en variabel. (1p)
- e) Föreslå en lösning på hur man hanterar att någon av argumenten/paramterarna är negativa. (2p)
- f) Finns det några värden på parametrarna när funktionen inte funkar alls? (1p)

Totalt: 7p

LÖSNINGSFÖRSLAG

a)

```
int nr_of_groups(int students, int students_per_groups) {  
    // dela antalet studenter med antalet studenter per  
    // grupp. Eventuell rest "skiter vi i".  
    int full_groups = students / students_per_groups;  
  
    // Ta reda på resten om man delar antalet studenter  
    // med antalet studenter per grupp.  
    int remaining = students % students_per_groups;  
  
    // Om resten är mindre än 2 returnerar vi antalet  
    // grupper... med antagandet att vi slänger in de
```

```

    resterande 0 eller 1 studenterna i befintliga grupp
    if (remaining < 2) {
        return full_groups;
    }

    // Om vi har 2 eller fler studenter utan grupp,
    skapar vi en ny (+1)
    return full_groups+1;
}

```

b) Funktionen avslutas – inget mer kod i funktionen körs. Om return har ett argument (t ex return 2) kan den anropande funktionen spara undan det returnerade värdet (2).

c)

$7/4 \Rightarrow 1$ Dela ut 7 hela glassar till 4 barn på ett rättvist sätt. "Självklart" är det mest rättvist att dela ut en till varje barn och slänga 3 glassar.

$7\%4 \Rightarrow 3$... de tre glassarna från ovanstående uträkning

3 är ju inte mindre än 2 så vi går inte "in i if-satsen".

Alltså returneras 1+1, d v s 2

d) `int groups = nr_of_groups(4,7);`

```

e) if ( (students < 0) || (students_per_groups<0) ) {
    return -1;
}

```

f) Om `students_per_groups` är 0 blir det division med noll... aj aj aj

5 (*struct, pointer*)

a) Skapa en struct, `monster`, som kan lagra följande värden separat (3p):

- position i x-led
- position i y-led
- slagkraft (används för att räkna ut vilken skada monstret kan tillfoga)
- hälsa

Motivera dina val av typer.

b) Använd typedef för att göra structen lite enklare att använda (1p).

c) Skriv en funktion, `set_monster_values`, som kan sätta alla ingående värden/variabler i en sådan struct (vilken som helst). Efter funktionen anropats skall monster-variabelns ursprungliga värden ha ändrats. (5p)

d) Skriv en funktion, `init_monster`, som initierar en monster-variabels värden enligt följande (4p):

- position i x-led - skall sättas till 100
- position i y-led - skall sättas till 100
- slagkraft - skall sättas till 50
- hälsa - skall sättas till 100

e) Använd funktionen `set_monster_values` för att implementera funktionen `init_monster` (3p).

d) Deklarera en array med plats för tio monster. (1p)

Totalt: 17p

LÖSNINGSFÖRSLAG

Se filen `monster.c`

6 (*dynamic memory*)

a) Skriv en funktionen, `new_monster`, som (8p):

- allokerar minne dynamisk (för en ny monster-variabel)
- initierar minnet (den nya monster-variabeln) med hjälp av funktionen, `init_monster`.
- returnerar en pekare till detta minne om allt gick bra.
- returnerar NULL om något gick fel.
- INGA utskrifter skall/bör/behöver göras i funktionen

OBS: funktionen skal kunna hantera att minnet tar slut på ett bra sätt. Lämpliga manualer finns i APPENDIX

b) Skriv en funktion som skriver ut ett monster (vilket som helst) (2p).

c) Skriv en funktion som skapar två monster med `new_monster` och skriver ut dessa (3p).

Totalt: 13p

LÖSNINGSFÖRSLAG

Se filen `monster.c`

7 (Array)

Skriv en funktion som skriver ut (med hjälp av printf) alla heltalsvärden (int) från en heltalsarray alternativt från en *pekare till int* (int*). Arrayen anges som parameter till funktionen tillsammans med antalet element i arrayen.

Prototypen för funktionen ser ut så här:

```
void print_int_array(int *array, int size);
```

OBS: funktionen måste hantera dålig indata Om den som använder er funktion anger en för stor storlek (size) jämfört med arrayens verkliga storlek kan ni lugnt låta användaren få skylla sig själv

Totalt: 4p

LÖSNINGSFÖRSLAG

Se filen array.c

99 (gör inte denna uppgift – bara på skoj)

Betrakta följande kod:

```
char *name = "fc liverpool";
int i ;
int j = 0;
for (int i=0;i<strlen(name); i++) {
    if (i>12) {
        j++;
    } else if (! (i>12)) {
        j--;
    } else {
        j=j=j;
    }
    if (i+(j=13)) { printf ("-"); }
    for (int j=i%7;j<i%13; i++) {
        printf ("%c", name[j+i]);
    }
}
```

Är den vettig? (0p)

APPENDIX - *Dynamic memory allocation*

```
void *malloc(size_t size);  
void free(void *ptr);  
void *calloc(size_t nmemb, size_t size);  
void *realloc(void *ptr, size_t size);
```

DESCRIPTION

The `malloc()` function allocates `size` bytes and returns a pointer to the allocated memory. The memory is not initialized. If `size` is 0, then `malloc()` returns either `NULL`, or a unique pointer value that can later be successfully passed to `free()`.

The `free()` function frees the memory space pointed to by `ptr`, which must have been returned by a previous call to `malloc()`, `calloc()`, or `realloc()`. Otherwise, or if `free(ptr)` has already been called before, undefined behavior occurs. If `ptr` is `NULL`, no operation is performed.

The `calloc()` function allocates memory for an array of `nmemb` elements of `size` bytes each and returns a pointer to the allocated memory. The memory is set to zero. If `nmemb` or `size` is 0, then `calloc()` returns either `NULL`, or a unique pointer value that can later be successfully passed to `free()`.

The `realloc()` function changes the size of the memory block pointed to by `ptr` to `size` bytes. The contents will be unchanged in the range from the start of the region up to the minimum of the old and new sizes.

If the new size is larger than the old size, the added memory will not be initialized. If `ptr` is `NULL`, then the call is equivalent to `malloc(size)`, for all values of `size`; if `size` is equal to zero, and `ptr` is not `NULL`, then the call is equivalent to `free(ptr)`. Unless `ptr` is `NULL`, it must have been returned by an earlier call to `malloc()`, `calloc()` or `realloc()`. If the area pointed to was moved, a `free(ptr)` is done.

RETURN VALUE

The `malloc()` and `calloc()` functions return a pointer to the allocated memory, which is suitably aligned or any built-in type. On error, these functions return `NULL`. `NULL` may also be returned by a successful call to `malloc()` with a size of zero, or by a successful call to `calloc()` with `nmemb` or `size` equal to zero.

The `free()` function returns no value.

The `realloc()` function returns a pointer to the newly allocated memory, which is suitably aligned for any built-in type and may be different from `ptr`, or `NULL` if the request fails. If `size` was equal to 0, either `NULL` or a pointer suitable to be passed to `free()` is returned. If `realloc()` fails, the original block is left untouched; it is not freed or moved.