

Screencast

(En första introduktion till) Minneshantering

Översikt

- Tillsvidare kan vi tänka oss att minnet i C består av tre delar:
 1. *Programminnet* – där programmets kod lagras, och alla konstanter
 2. *Stacken* – som lagrar allt data som finns i lokala variabler i funktioner
 3. *Heapen* – som lagrar allt data som skall ”överleva en funktion”
- Programminnet är ett konstant minne som inte får ändras. Om man försöker ändra på en konstant kan man räkna med att programmet kraschar.
- *Stacken* lagrar allt minne som finns i lokala variabler. Om en heltalsvariabel lagras i 8 bytes kräver en funktion f med en variabel `int x` minst 8 bytes av stackminne varje gång f anropas. (Plus en del extra minne för metadata om f , men vi bortser från det här.)
- *Stackens* minne hanteras *automatiskt*, d.v.s. vid anrop till f kommer 8 bytes att allokeras på stacken för att hålla värdet i variabeln x . När f returnerar avallokeras detta minne automatiskt i och med att x försvinner.
- Allokering och avallokering på stacken är väldigt tidseffektivt. Stacken är oftast en ganska liten del av minnet eftersom de flesta program ryms på en relativt grund stack.

- Allt minne som inte går åt att hålla programmet eller stacken kallas *heapen*. Till skillnad från stackminnet måste heapminnet hanteras manuellt i C. Effektivt betyder det:
 1. Varje allokering och avallokering sker explicit
 2. Vid varje allokering måste storleken på det efterfrågade minnet anges explicit

Stackens minne är kortlivat men data i heapminnet kan leva länge.

- Allokering på heapen sker med funktionen `malloc` (se även `calloc` och `realloc`). För att allokera 8 bytes på heapen skriver man således:

```
... = malloc(8);
```

Malloc allokera minne på heapen av efterfrågad storlek och returnerar en pekare till detta minne.

- Avallokering sker med funktionen `free` som anropas med adressen till det utrymme man vill frigöra.

Fråga 1: Var sparas värdet i variabeln `tal`?

```
1  int main(void) {  
2      int tal = 3;  
3      ...  
4      tal = ...;  
5  }
```

1. I programminnet
2. På stacken
3. På heapen

Fråga 1: Var sparas värdet i variabeln `tal`?

```
1  int main(void) {  
2      int tal = 3;  
3      ...  
4      tal = ...;  
5  }
```

1. I programminnet
2. På stacken
3. På heapen

Rätt svar På stacken. `tal` är en helt vanlig lokal variabel. Konstanten 3 sparas dock i programminnet och vid körning tilldelas `tal` från denna plats i programminnet vid start av `main` – då kopieras en 3:a in i `tal`.

I programminnet är inte rätt eftersom `tal` är en variabel och inte en konstant. Dess initiala värde 3 sparas dock i programminnet.

På heapen är inte rätt eftersom det inte finns några anrop till `malloc`.

Fråga 2: Var sparas värdet i variabeln `str`?

```
1  int main(void) {  
2      char *str = "Hello";  
3      ...  
4      str = ...;  
5  }
```

1. I programminnet
2. På stacken
3. På heapen

Fråga 2: Var sparas värdet i variabeln `str`?

```
1  int main(void) {  
2      char *str = "Hello";  
3      ...  
4      str = ...;  
5  }
```

1. I programminnet
2. På stacken
3. På heapen

Rätt svar På stacken. Samma logik som i föregående exempel. Strängen "Hello" sparas i programminnet och vid körning tilldelas `str` adressen till denna plats i programminnet vid start av `main`.

Fråga 3: Var sparas strängen som variabeln `str` pekar på?

```
1  int main(void) {  
2      char *str = "Hello";  
3      ...  
4      str = ...;  
5  }
```

1. I programminnet
2. På stacken
3. På heapen

Fråga 3: Var sparas strängen som variabeln `str` pekar på?

```
1  int main(void) {  
2      char *str = "Hello";  
3      ...  
4      str = ...;  
5  }
```

1. I programminnet
2. På stacken
3. På heapen

Rätt svar På stacken. Samma logik som i föregående exempel. Strängen "Hello" sparas i programminnet och vid körning tilldelas `str` adressen till denna plats i programminnet vid start av `main`.

Fråga 3: Var sparas strängen som variabeln `str` pekar på?

```
1  int main(void) {  
2      char *str = "Hello";  
3      ...  
4  }
```

1. I programminnet
2. På stacken
3. På heapen

Fråga 3: Var sparas strängen som variabeln `str` pekar på?

```
1  int main(void) {  
2      char *str = "Hello";  
3      ...  
4  }
```

1. I programminnet
2. På stacken
3. På heapen

Rätt svar I programminnet. Det är en konstant (en strängliteral) som är en del av källkoden. När programmet körs laddas programmet in i minnet, inklusive strängen "Hello". Varje byte av detta minne har en adress, och låt A vara adressen till strängen "Hello". När `str` initieras sparas adressen A *på stacken* i den plats som `str` avser, men *strängen* som `str` *pekar ut* ligger fortfarande i programminnet.

Pröva gärna att göra `*str = 'x';` vilket går till adressen A och skriver tecknet 'x'. Detta ändrar i det faktiska C-programmet vilket i de flesta operativsystem inte tillåter, utan "skjuter ned" programmet.

Fråga 4: Var sparas strängen i variabeln `str`?

```
1  int main(void) {  
2      char str[] = "Hello";  
3      ...  
4  }
```

1. I programminnet
2. På stacken
3. På heapen

Fråga 4: Var sparas strängen i variabeln `str`?

```
1  int main(void) {  
2      char str[] = "Hello";  
3      ...  
4  }
```

1. I programminnet
2. På stacken
3. På heapen

Rätt svar På stacken. Detta är en slamkrypare, men en viktig sådan. I fråga 3 var `str` en pekare till en sträng, men här är `str` en array vars innehåll är tecknen i "Hello" plus ett nulltecken. "Hello" är som vanligt en strängkonstant som sparas i programminnet, men vid initiering av `str` kopieras inte en adress in i variabeln utan samtliga sex tecken. Precis som tidigare är `str` en stackvariabel och dess innehåll ligger på stacken.

Fråga 5: Var sparas strängen i variabeln `str`?

Funktionen `strlen` returnerar antalet tecken i en sträng.

Funktionen `strcpy` kopierar innehållet i `hello` till `str`.

```
1  int main(void) {  
2      char *hello = "Hello";  
3      char *str = malloc(strlen(hello)+1);  
4      strcpy(str, hello);  
5      ...  
6  }
```

1. I programminnet
2. På stacken
3. På heapen

Fråga 5: Var sparas strängen i variabeln `str`?

Funktionen `strlen` returnerar antalet tecken i en sträng.

Funktionen `strcpy` kopierar innehållet i `hello` till `str`.

```
1  int main(void) {  
2      char *hello = "Hello";  
3      char *str = malloc(strlen(hello)+1);  
4      strcpy(str, hello);  
5      ...  
6  }
```

1. I programminnet
2. På stacken
3. På heapen

Rätt svar: På heapen. Båda variablerna `hello` och `str` är stackvariabler och deras innehåll ligger på stacken. I detta fall är båda pekare. `hello` pekar ut en sträng i programminnet (enligt logik från tidigare fråga) medan `str` pekar på *explicit* allokerat minne med hjälp av `malloc` som alltså ligger på heapen. Funktinen `strcpy` kopierar innehållet i `hello` till `str`, d.v.s. från programminnet till heapen.

- Fråga 5 visar hur delikat manuell minneshantering kan vara. strlen returnerar antalet tecken i en sträng, men eftersom vi också måste ha plats för nulltecknet behöver vi lägga på ytterligare 1 tecken. Glömmer vi det kan strcpy komma att skriva utanför det allokerade minnet och förstöra det data som kommer "efter".
- Det är enkelt att glömma att avallokera minne med free och är en av de vanligaste felen C-programmerare begår, oavsett erfarenhetsnivå.
- I stora program där många minnesplatser pekas ut från olika ställen är det väldigt svårt att veta inte bara *när* det är säkert att avallokera data men också *vem* som är ansvarig för när det skall göras.
- När man avallokerar minne nollställs det inte utan bara flaggas som tillgängligt. Eftersom malloc inte heller nollställer minne vid allokering betyder detta effektivt sett att minnet kan innehålla en massa skräp.