Strängar i C

- Många programmeringsspråk har en speciell datatyp för strängar – dock inte C
- I C är en sträng helt enkelt en räcka av tecken (array of **`char`**)

  => Ett klumpigt sätt att skapa en sträng är att sätta in bokstäverna “en och en” i en räcka:

```
char myString[10];
myString[0] = 'H';
mystring[1] = 'e';
...
```

Strängar i C

- En sträng kan också initialiseras på ett enklare sätt:

```
char myString[] = "Hello World!";
```

- En strängvariabel kan skrivas ut med **%s** i en **printf()**-sats:

```
printf("This is my string: %s\n", myString);
```

- En sträng kan läsas in från användaren med **scanf**. I detta fall behövs **inte** något **&**-tecken framför variabeln:

```
scanf("%s", myString);
```

Strängar i datorns minne

- En sträng är en array och lagras därmed på samma sätt som en vanlig räcka i datorns minne
- Jobbigt att hålla reda på när en sträng tar slut, dvs. hur många positioner det finns i räckan

`word[0]`	`'H'`
`word[1]`	`'e'`
`word[2]`	`'l'`
`word[3]`	`'l'`
`word[4]`	`'o'`
`word[5]`	`'!'`
`word[6]`	`'\0'`

=> för att slippa lagra denna information används en “null character” `'\0'` för att indikera att strängen tagit slut

- Sätts in *automatiskt* om strängen initaliseras samtidigt som den deklarareras

Strängar och funktioner

- En sträng är en räcka, så funktioner fungerar på motsvarande sätt som för andra typer av räckor:

```
int countChar(char searchString[], char toFind)
{
    int noOfOccurences = 0;
    // Search for the # of occurences
    // of toFind in searchString...

    return noOfOccurences;
}

char myString[] = "I'm a great programmer";
int result = countChar(myString, 'm');
// result should now be 3
```

Funktioner för stränghantering

- Metoden att läsa in strängar med **`scanf()`** fungerar om man endast vill läsa in ett ord åt gången
- **`scanf()`** tillåter att man t.ex. använder mellanslag för att separera “input items”
  => att läsa in texten **`Hello World`** som en enda sträng fungerar **inte**
- Vanligt att man vill *jämföra* innehållet i två strängar med varandra – men en jämförelse (**`string1 == string2`**) fungerar **inte**
- Exempel på andra vanligt förekommande uppgifter:
  Sök efter en *delsträng* inne i en annan sträng,
  kombinera två strängar

Funktioner för stränghantering

- C:s standardbibliotek **`stdio`** och **`string`** innehåller funktioner för stränghantering
- Appendix B i kursboken räknar upp de olika funktioner som finns tillgängliga
- Gemensamt för alla biblioteksfunktioner som hanterar strängar är att de egentligen använder sig av *pekare* till strängar (oviktigt i detta skede)

Funktioner för stränghantering

- **`fgets(s, n, stdin)`** – Läser in högst **`n-1`** bokstäver som lagras i strängen **`s`** från tangentbordet (=**`stdin`**)
- **`getchar()`** för att läsa in endast ett tecken – alternativ till **`scanf("%c", ch)`**
- **`strncat(s1,s2,n)`** – Konkatenerar (lägger till) högst **`n`** tecken från strängen **`s2`** efter strängen **`s1`**
- **`strcmp(s1,s2)`** – Jämför strängen **`s1`** och strängen **`s2`**
- **`strncpy(s1,s2,n)`** – Kopierar högst **`n`** tecken från **`s2`** till **`s1`**
- **`strstr(s1,s2)`** – Söker efter **`s2`** i **`s1`**
- **`strlen(s1)`** – Kontrollerar längden på **`s1`**

Funktioner för stränghantering

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

int main(void) {
    char row1[100] = ""; // Init to empty string
    char row2[100] = ""; // Ditto

    printf("Enter two sentences!\n");
    fgets(row1, 100, stdin); // Read at most 99 chars,
                             // adds '\0' to end of string
    fgets(row2, 100, stdin); // stdin means 'Standard input'

    if (strcmp(row1, row2) == 0) {
        printf("The two sentences are the same!\n");
    }

    return 0;
}
```

Funktioner för stränghantering

- Tillgång till inbyggda strängfunktioner förenklar programmerarens liv
- Ändå nyttigt att studera hur motsvarande funktionalitet skulle kunna implementeras utan tillgång till standardbiblioteken
- Exempel: **`equalstrings.c`** i modellösningsmappen
  - Jämför innehållet i två strängar
  - Endast true/false som resultat
  - Hur kan vi modifiera funktionen för lexikografisk (alfabetisk) jämförelse av strängar?

Strängar och structs

- En struct kan också innehålla räckor – och därmed även strängar

```
struct textualDate {
    char weekday[10];
    int day;
    char month[10];
    int year;
};

struct textualDate today = { .weekday = "Wednesday",
                             .day = 19,
                             .month="October",
                             .year = 2011 };
```

Strängar och räckor

- Kan lagra många strängar i en räcka
  => Tvådimensionell räcka
- Måste bestämma maxlängd för strängarna då räckan skapas

```
char weekdays[7][10] = {"monday", "tuesday",
                        "wednesday", "thursday",
                        "friday", "saturday", "sunday"};

printf("%s\n", weekdays[3]); => output is "thursday"
```