Dynamisk minneshantering

- När en variabel deklareras reserveras exakt så mycket minne som behövs för denna variabel – t.ex. 1 byte för en **`char`**, 4 bytes för en **`int`**, 80 bytes för en räcka med 20 **`int`**...
- Tre sätt att deklarera en räcka:

  a) Skapa en räcka där antalet element är fastslaget vid kompileringen: **`int myArray[20];`**

  b) Skapa en räcka där antalet element är fastslaget när räckan skapas, med hjälp av 'variable-length arrays'

```
int sizeOfArray = determineNeededLength();
int myArray[sizeOfArray];
```

  c) Allokera minne *dynamiskt* när programmet körs, förändra storleken på räckan efter behov

Dynamisk minneshantering: malloc

- Funktionen **`malloc()`** används för att reservera minne
- **`malloc()`** tar som parameter *antalet bytes* som skall reserveras, och returnerar en *pekare* till den första adressen i detta minnesområde
- För att med säkerhet veta hur många bytes som behövs för att t.ex. reservera minne för ett visst antal integers kan funktionen **`sizeof()`** användas:

```
malloc(sizeof(int)*10); // reserves memory
                        // for 10 integers
```

Dynamisk minneshantering: malloc

- **`malloc`** returnerar en speciell typ av pekare:
  en pekare till **`void`**
- Kan här tolkas som “en pekare till vad som helst”.
- Programmeraren måste sedan själv omvandla denna pekare till korrekt datatyp genom en *type cast:*

```
int* pointer =
    (int*) malloc(sizeof(int)*10);
```

- En minnesallokering kan misslyckas
  (dvs minnet kan ta slut!)
  I så fall returnerar **`malloc`** en “nullpekare” (**`NULL`**)

Dynamisk minnesallokering: malloc

```
#include <stdio.h>
#include <stdlib.h> // required for malloc()

int main(void) {
	int size = 0;
	int* arrayPointer = 0;
	printf("How many elements? ");
	scanf("%d", &size);
	arrayPointer = (int*) malloc(sizeof(int) * size);
	if (arrayPointer == NULL) // Or like this: if (!arrayPointer)
	{
		printf("Allocation failed! Out of memory?\n");
		return 0;
	}
	// Do something nice with your array
	return 0;
}
```

Dynamisk minneshantering: realloc

- **`realloc()`** används för att öka eller minska storleken på ett reserverat minnesområde:
- Liksom **`malloc()`** returnerar **`realloc()`** en nullpointer om minnesområdet inte kan allokeras
- **`realloc()`** kan välja att utöka det existerande minnesområdet, eller att allokera ett helt nytt område
  => vi kan **inte** utgå från att den gamla datan ligger kvar på samma plats i minnet efter ett **`realloc()`**-anrop

Dynamisk minnesallokering: realloc

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

int main(void) {
     int size = 10;
     int* arrayPointer = (int*) malloc(sizeof(int)*size);
     // ...
     // Double the space
     arrayPointer = (int*) realloc(arrayPointer,
                                   sizeof(int)*(size*2));
     // ...
     // Make the array smaller
     arrayPointer = (int*) realloc(arrayPointer,
                                   sizeof(int)*(size/4));
     // ...
     return 0;
}
```

Dynamisk minneshantering: Minnesläckor

- Vad blir effekten av nedanstående kod?

```
arrayPointer = (int*) malloc(sizeof(int)*size);
…
// Double the space
arrayPointer = (int*) malloc(sizeof(int)*size*2);
```

- Det andra **`malloc()`**-anropet allokerar nytt minne
- Det tidigare minnesområdet är fortfarande reserverat för programmets räkning, men det finns inte längre någon pekare till detta minnesblock

  => Vi har åstadkommit en *minnesläcka*

Minnesläcka i en hiss (pseudokod)

- When a button is pressed:
- Get some memory, which will be used to remember the floor number
- Put the floor number into the memory
- Are we already on the target floor?
  - If so, we have nothing to do: finished!
- Otherwise:
  - Wait until the lift is idle
  - Go to the required floor
  - Release the memory we used to remember the floor number

Dynamisk minneshantering: free

- I de flesta fall – **men inte alltid** – kommer det dynamiskt reserverade minnet att frigöras när programmet avslutas
- Säkrast att alltid frigöra dynamiskt minne manuellt när det inte längre behövs
- Minnesläckor i t.ex. operativsystem eller inbyggda system är speciellt allvarliga – varför?

Dynamisk minneshantering: free

- För att frigöra dynamiskt allokerat minne används funktionen **`free()`:**

```
arrayPointer =
        (int*)malloc(sizeof(int)*size);
…
// Release the memory
free(arrayPointer);
```

- Vanliga misstag:
  - Försök att frigöra ett minnesområde som inte har allokerats dynamiskt
  - Försök att frigöra samma minnesområde två gånger