Algorithmische Mathematik I

Wintersemester 2009/2010
Prof. Dr. Mario Bebendorf
Dr. Jan Hamaekers

universitätbonn

Programmierhinweise: Speicherverwaltung

Im Folgenden finden Sie einführende Beispiele in die Speicherverwaltung von `C` mithilfe von `calloc()` und `free()` bzw. von `C++` mit `vector`.

Dynamische Array-Verwaltung in `C`

In `C` wird Speicher „roh“ als `void*` vom System angefordert, wobei man in der Regel die Größe in Vielfachen von Bytes angibt. Das geht mit

```
void *calloc(size_t n, size_t size);
```

Diese Funktion „alloziiert“ Speicher für ein Array mit `n` Elementen, von denen jedes `size` Bytes groß ist. Mit `sizeof(int)` kann man beispielsweise die Größe in Bytes eines `int` herausfinden.

Den so gewonnenen Speicher muss man dann auf `int*` „casten“, damit `C` weiß, dass es sich um Integerarray handelt.[1]Wenn man das Array nicht mehr braucht, muß man es „von Hand“ freigeben mit `free()`. Beispiel:

```
#include <stdlib.h>

void eineFunction( const int* A, unsigned int n )
{
    int a = A[2];    /* Zugriff auf Elemente */
    int b = A[3];
    int* andererVektor = (int*)calloc( n+3, sizeof(int) );
    andererVektor[n] = A[0];
    /* irgendwas tun ... */
    free( andererVektor ); /* freigeben nicht vergessen! */
}

int main(int argc, char** argv)
{
    unsigned int n = 16;
    int* A = (int*)calloc( n, sizeof(int) );
    /* Bedienung genauso, wie bei den bekannten Arrays. */
    A[0] = 4;
    A[1] = -1;
    eineFunction(A, n);
    free(A); /* freigeben nicht vergessen! */
    return 0;
}
```

Das Beispiel kann man mit ‘`gcc datei.c`’ übersetzen.

[1]Eigentlich heißt `int*` „Zeiger auf `int`“. `C` speichert Arrays grundsätzlich so, daß man einen Zeiger auf das erste Element bekommt.

Dynamische Array-Verwaltung in C++

C++ bietet fertige Arrays, bei denen man nicht „von Hand“ den Speicher freigeben muß, was erfahrungsgemäß viele Fehler vermeidet. Dynamische Speicherverwaltung mit diesen Arrays in C++ ist im folgenden Codefragment erklärt.

```
#include <vector>

/* diese Zeile kann man weglassen, dann muss man
 * immer 'std::vector' schreiben. */
using namespace std;

/* vector<int>& heisst: referenz auf vector. */
void eineFunction( const vector<int>& A )
{
    int a = A[2];   // Zugriff auf Elemente
    int b = A[3];
    unsigned int n = A.size(); // hole die Groesse des Arrays
    vector<int> andererVektor( A.size() + 3 );
    /* irgendwas tun ... */
    andererVektor[A.size()] = A[0];
    // bei Verlassen der Funktion wird 'andererVektor' freigegeben.
}

int main(int argc, char** argv)
{
    unsigned int n = 16;
    vector<int> A(n);
    // Bedienung genauso, wie bei den bekannten Arrays.
    A[0] = 4;
    A[1] = -1;
    eineFunction(A);
    return 0;
}
```

Das Beispiel kann man mit ‘g++ datei.cc’ übersetzen.
Für weitere Hinweise sei auf die Literatur verwiesen, siehe die Liste auf der Vorlesungshomepage http://wissrech.ins.uni-bonn.de/teaching/algmath/ws09_i.