Memory Management and Optimizations

Seminar „Effiziente Programmierung“

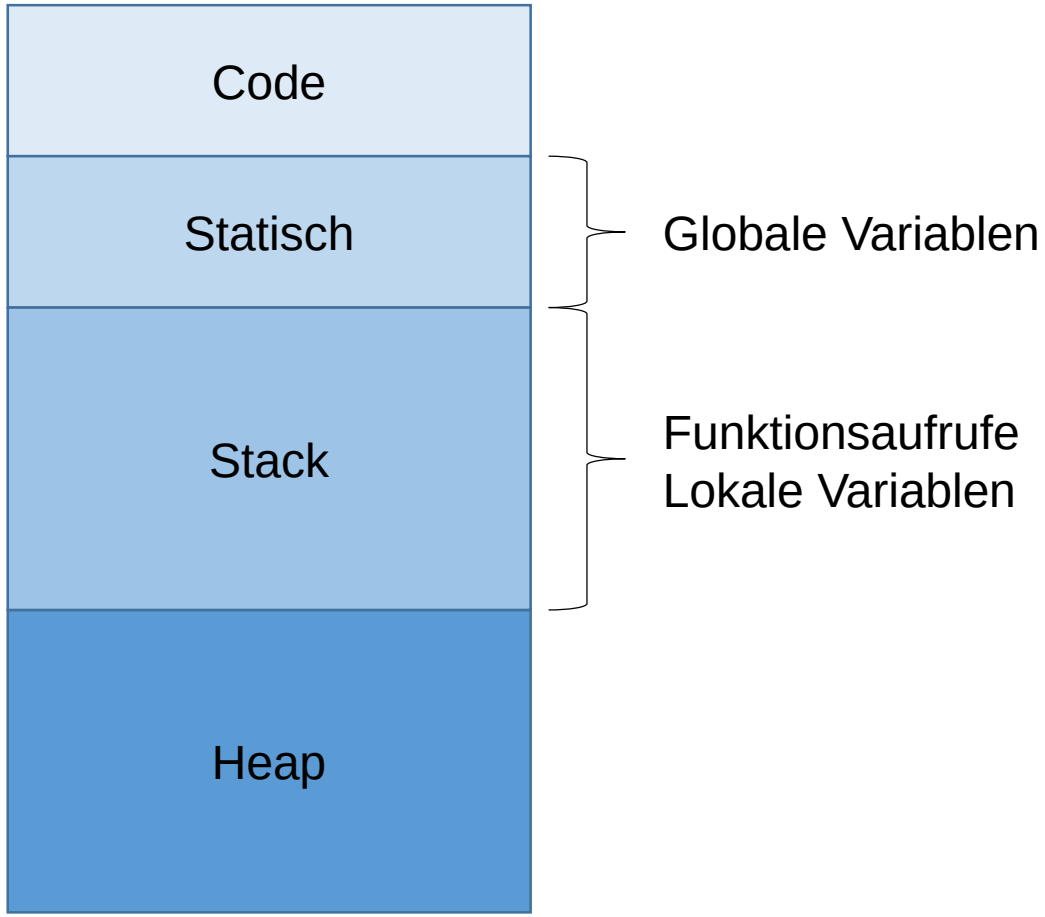
Tim Wischhof

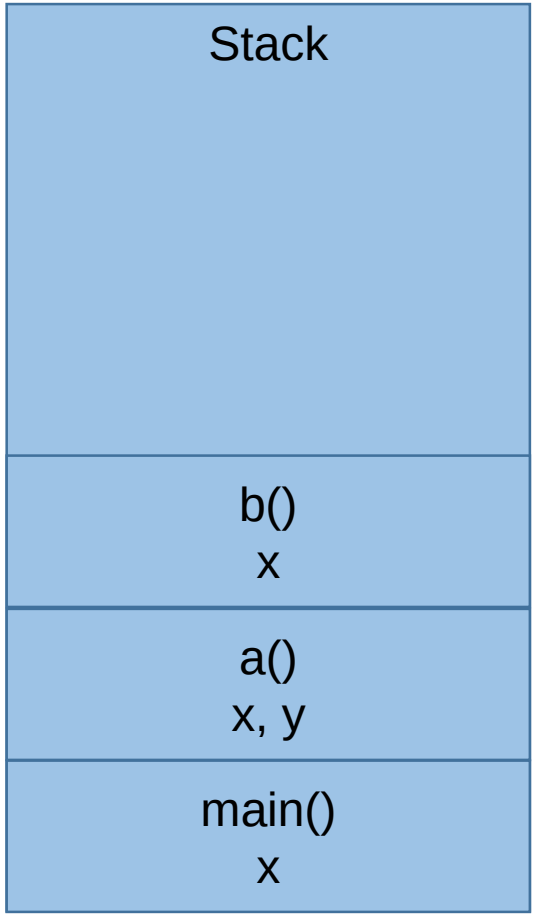
2017-12-21

Agenda

1. Statische Speicherverwaltung
2. Dynamische Speicherverwaltung
3. Speicheroptimierung

Statische Speicherverwaltung






```
1   int result;
2   int b(int x)
3   {
4       return x + 1;
5   }
6   int a(int x)
7   {
8       int y = x + 1;
9       return b(y);
10  }
11  int main()
12  {
13      int x = 0;
14      result = a(x);
15  }
```

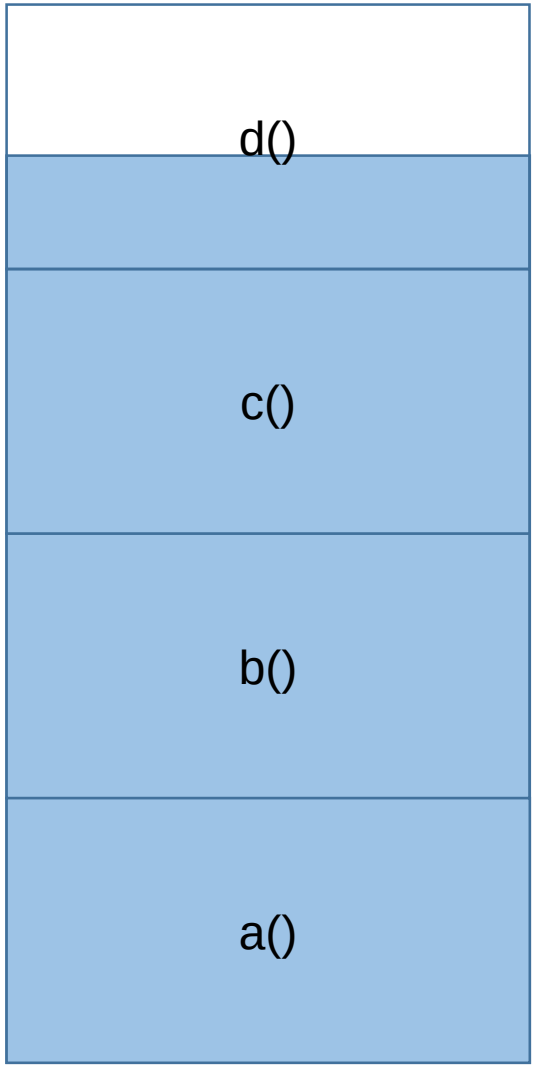
Stack

- Zu Programmstart wird eine bestimmte Menge an Speicher allokiert
  - Dieser Speicher ist während der Laufzeit nicht erweiterbar

- Die Erstellung der Stack Frames geschieht zur Laufzeit
  - Passiert automatisch durch die CPU
  - Frames enthalten neben Variablen auch die Aufrufadresse der Methode
  - Größe zur Compilezeit unbekannt

- Stack Frames können Größe des Stacks überschreiten
  - ➢ Stack Overflow

Stack Overflow




```c
int stack_overflow()
{
    return stack_overflow();
}
```

```c
int stack_overflow2()
{
    int x[1000000];
    for(int i = 0; i < sizeof(x); i++)
    {
        printf("%d\n",x[i]);
    }
}
```

Stack: Einschränkungen

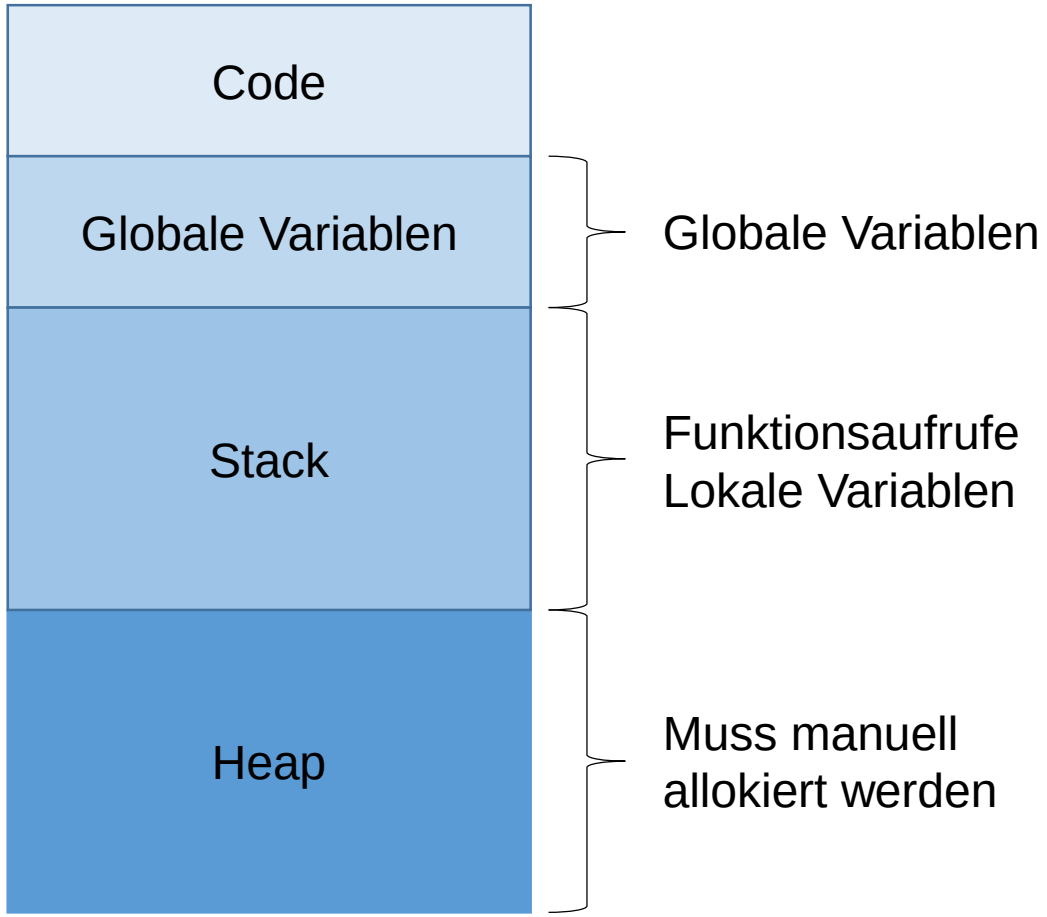
- Der Speicher kann zur Laufzeit nicht erweitert werden
  - Menge des nutzbaren Speichers ist begrenzt

- Die Größe aller Variablen muss zur Compilezeit feststehen
  - Dynamische Größen sind nicht möglich

- Die Lebensdauer von Variablen kann nicht beeinflusst werden
  - Variablen leben nur so lange wie ihre Methoden

➢ Heap

Dynamische Speicherverwaltung



Dynamischer Speicher / Heap

- Speicher kann während der Laufzeit allokiert werden
  - Größe des Heaps kann variieren
- Größe von Variablen kann zur Laufzeit angepasst werden
- Programmierer entscheidet über die Lebensdauer von Variablen
  - Speicher muss manuell freigegeben werden
- Heap kann potentiell wachsen, bis man den gesamten Speicherplatz erschöpft
- Zugriff langsamer als beim Stack

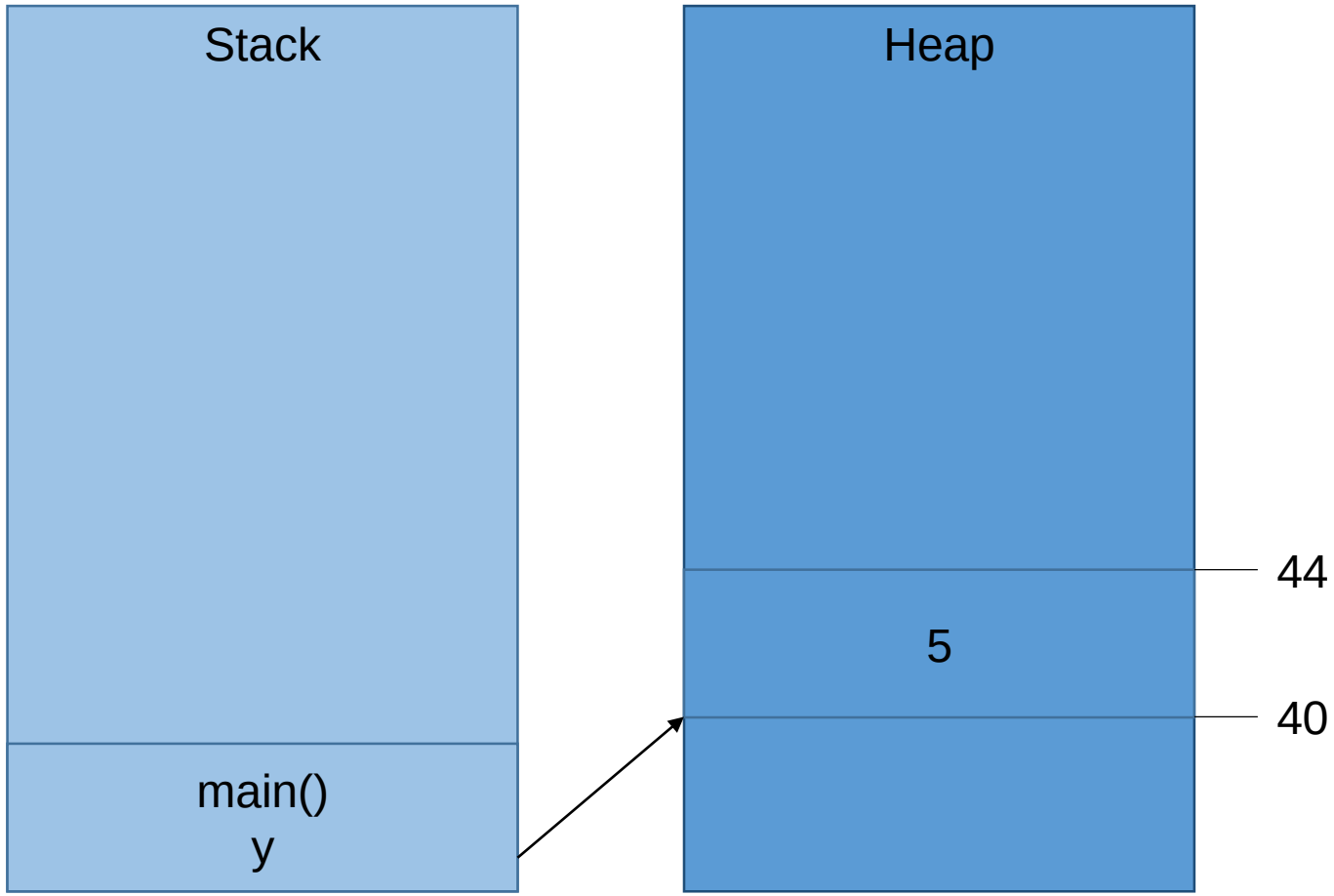
➢ Kann als großer, frei nutzbarer Speicher gesehen werden

Dynamische Speicherverwaltung in C

- Speicher kann mithilfe der Funktion „malloc“ auf dem Heap allokiert werden
  - void* malloc(size_t size)
  - Reserviert freien Speicher der Größe size
  - Returnt Pointer auf Startadresse des Blocks
  - Allokierter Speicher muss durch „free“ wieder freigegeben werden

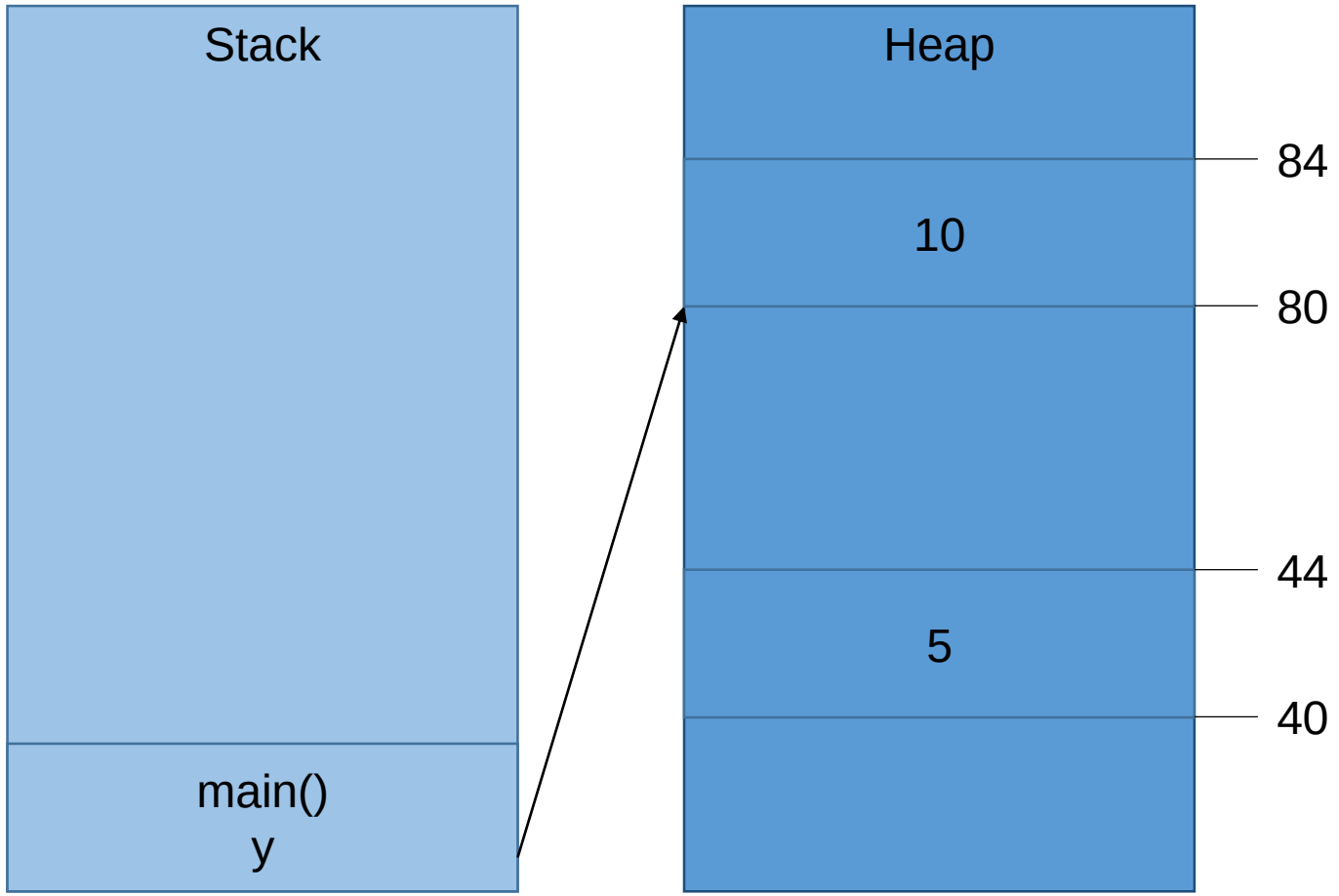
Beispiel: malloc

```
1  int main()
2  {
3      int *y;
4      y = malloc(sizeof(int));
5      *y = 5;
6  }
```



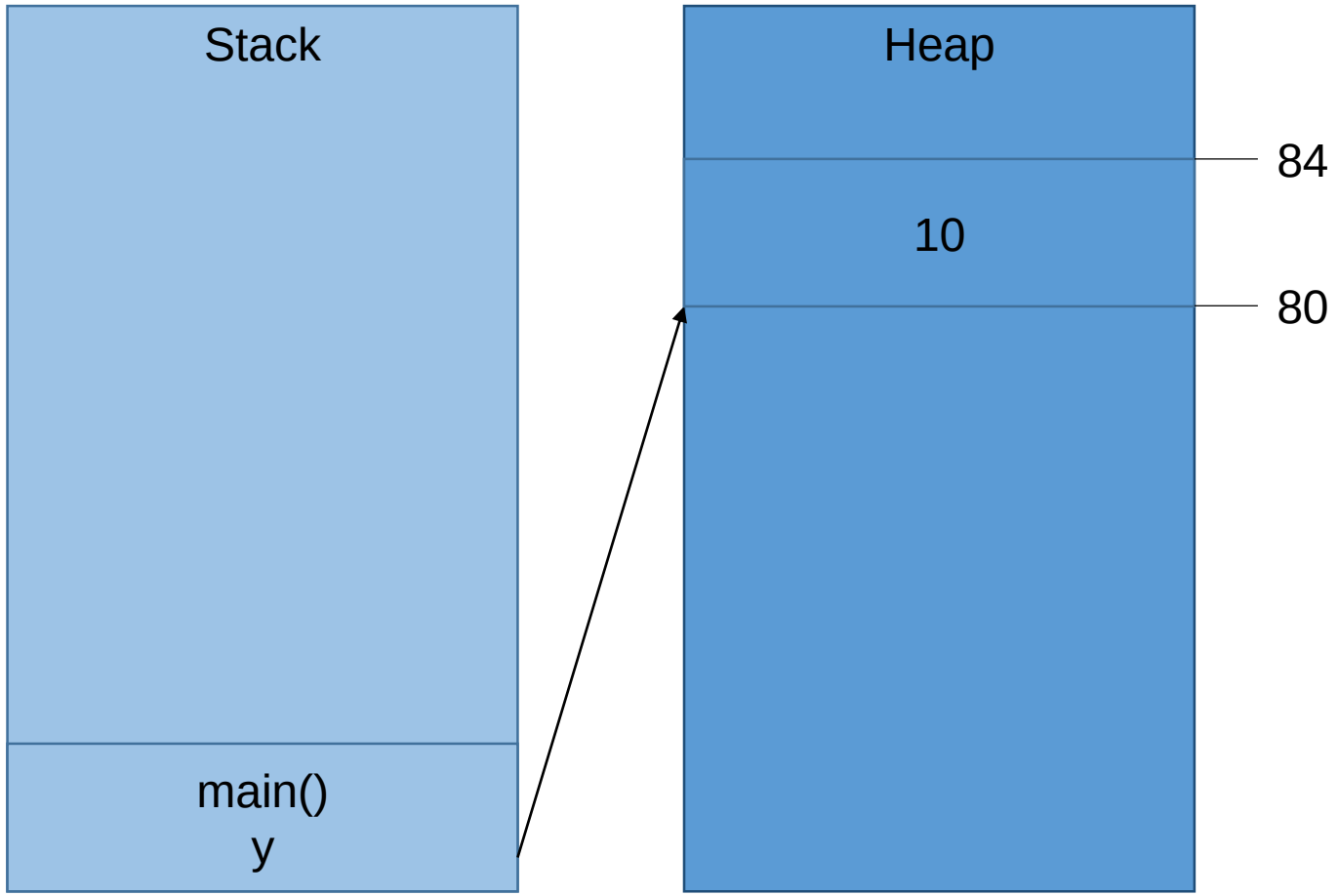
Beispiel: malloc 2

```
1  int main()
2  {
3      int *y;
4      y = malloc(sizeof(int));
5      *y = 5;
6
7      y = malloc(sizeof(int)):
8      *y = 10;
9  }
```



Beispiel: malloc 3

```
1   int main()
2   {
3       int *y;
4       y = malloc(sizeof(int));
5       *y = 5;
6       free(y);
7
8       y = malloc(sizeof(int)):
9       *y = 10;
10  }
```

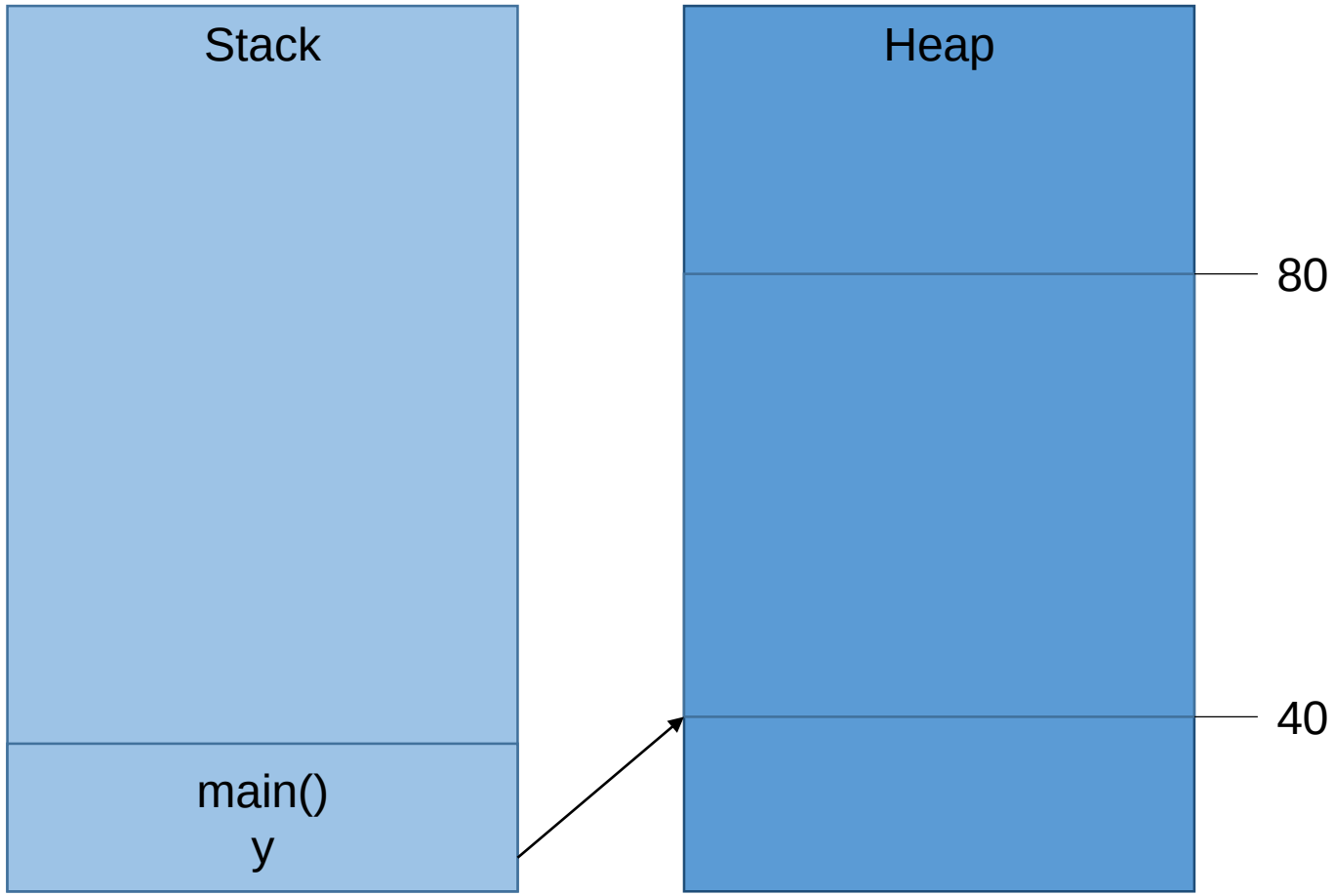


Beispiel: malloc 4

```c
int main()
{
    int *y;
    y = malloc(sizeof(int));
    *y = 5;
    free(y);

    y = malloc(10*sizeof(int));

}
```



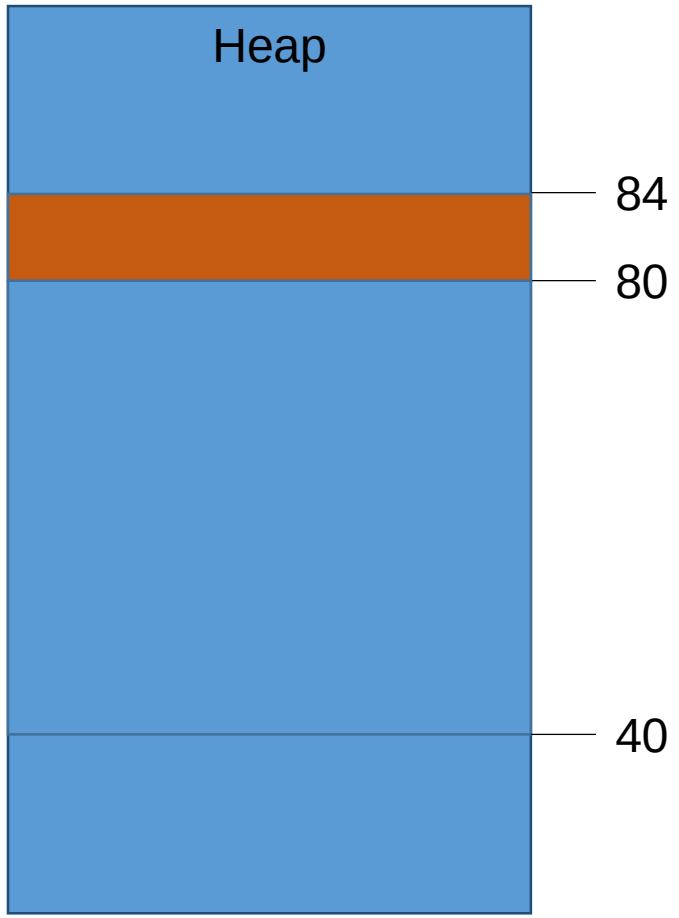
malloc, calloc, realloc, free

- malloc
  - reserviert Speicherbereich gegebener Größe
- calloc
  - initialisiert alle Bits mit 0
- realloc
  - reallokiert zuvor reservierten Speicher mit neuer Größe
- free
  - gibt zuvor allokierten Speicher wieder frei
- <stdlib.h>

Buffer Overflow

- Zugriff auf Speicher abseits der allokierten Adressen

```
1   int main()
2   {
3       int *y;
4       y = malloc(10*sizeof(int));
5       y[10] = 99;
6   }
```



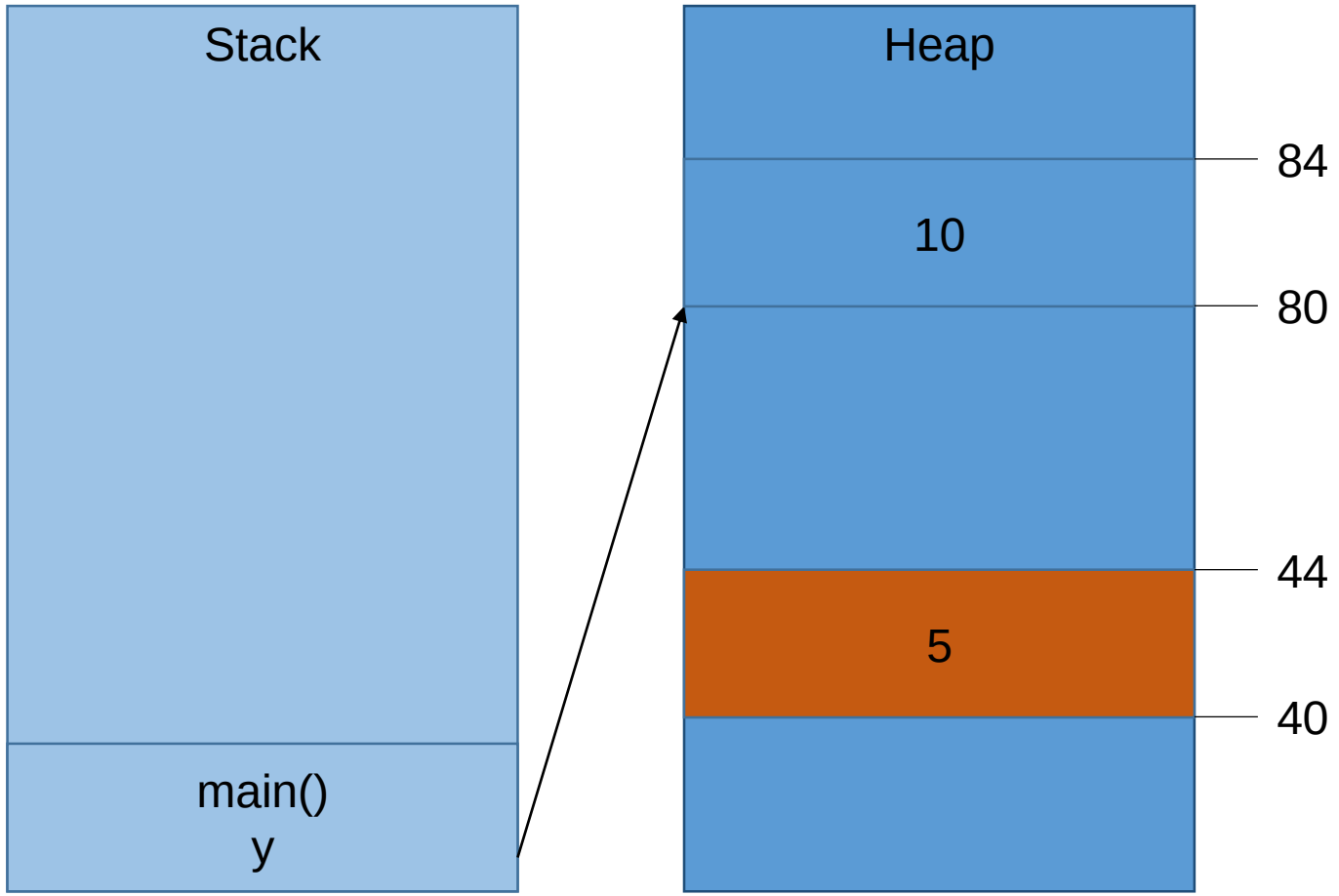
➢ Unvorhersehbare Ergebnisse

Memory Leak

- Speicher ist reserviert, wird aber nicht genutzt

```c
int main()
{
    int *y;
    y = malloc(sizeof(int));
    *y = 5;

    y = malloc(sizeof(int)):
    *y = 10;
}
```



Paging & Thrashing

- Paging: Teil des RAMs wird an Festplatte ausgelagert
  - ➢ Cache: Zeitliche und räumliche Lokalität

- Heap belegt zu viel Speicher (Memory Leak)
  - ➢ Konstantes Paging (Thrashing)

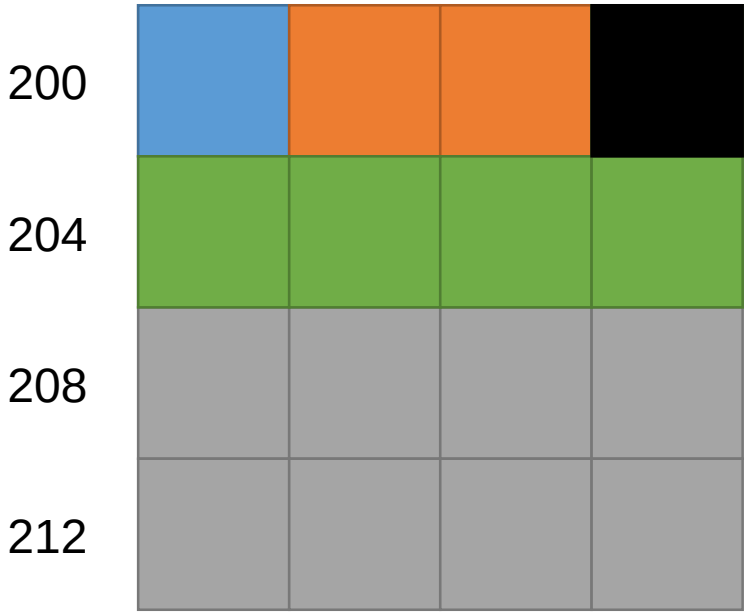
➢ Sehr zeitaufwending
  - ➢ Festplattenzugriffe ~100.000x langsamer als RAM

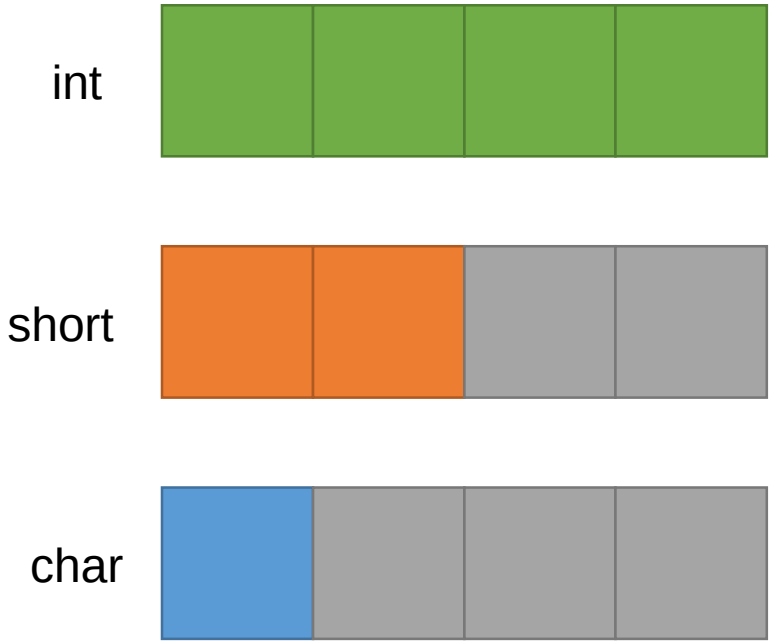
Zusammenfassung: Speicherverwaltung

- Dynamische Speicherverwaltung ist sinnvoll, wenn
  - wir viel Speicherplatz benötigen
  - Variablen über ihren Funktionsaufruf hinaus bestehen sollen
  - wir den Speicherbedarf zur Laufzeit anpassen möchten
- Dynamische Speicherverwaltung beinhaltet Risiken
  - Buffer Overflow
  - Memory Leak

Speicheroptimierung: Alignment 1

- Speicherzugriff auf Segmente bestimmter Größe
  - Typischerweise 4 oder 8 Byte



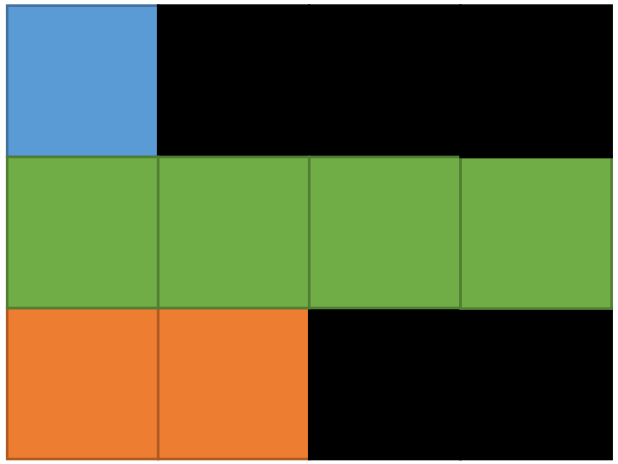


Speicheroptimierung: Alignment 2

- Compiler sorgt für Alignment der Daten

➢ Dies gilt nicht für structs

```
1   struct data
2   {
3       char x;
4       int y;
5       short z;
6   }
```

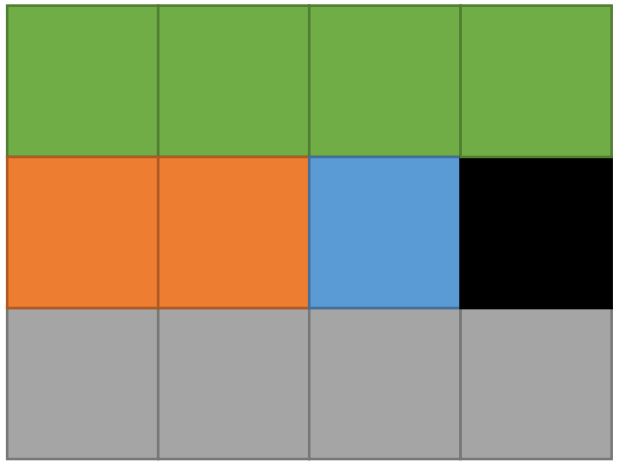


Speicheroptimierung: Alignment 3

- Structs durch Deklarationsfolge anpassen

➢ Größte Datentypen zuerst

```
1   struct data
2   {
3       int y;
4       short z;
5       char x;
6   }
```



Speicheroptimierung: Unions

- Unions speichern verschiedene Variablen am selben Ort

```c
union Data {
    float x;
    int y;
    char z;
};

int main()
{
    union Data data;

    data.x = 10.0;
    data.y = 100;
    data.z = 'q';

    printf("data.x: %e\n", data.x);
    printf("data.y: %d\n", data.y);
    printf("data.z: %c\n", data.z);

    return 0;
}
```

```
data.x: 1.583467e-043
data.y: 113
data.z: q
```

Speicheroptimierung: Compiler 1

- GCC bietet verschiedene Optimierungslevel:
  - O0
    - Default, keine Optimierung
  - O/O1
    - Optimierung mit Rücksicht auf Compilezeit
  - O2
    - Optimierung ohne große Space-Time Tradeoffs
  - O3
    - Optimierung mit Fokus auf Zeit

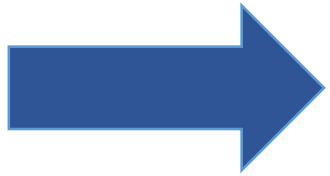
- Weitere Optimierungsmethoden:
  - Os
  - Ofast
  - Og

Speicheroptimierung: Compiler 2

- Alignment für Funktionen, Schleifen, Sprünge und Labels (O2/O3)
  - ➢ Folie 19 – 21

- Entfernen gemeinsamer Teilausdrücke (O2/O3/Os)

```
1    x = a * b + c;
2    y = a * b + d;
```

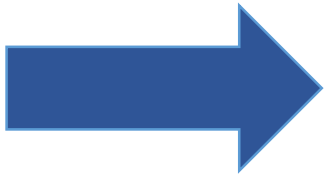


```
1    temp = a * b;
2    x = temp + c;
3    y = temp + d;
```

Speicheroptimierung: Compiler 2

- Inlining von Funktionen (O3)

```
1   int add(int x, int y)
2   {
3       return x + y;
4   }
5
6   int inlineFunction()
7   {
8       return add(1, 2) * add(3, 4);
9   }
```



```
1   int inlineFunction()
2   {
3       return (1 + 2) * (3 + 4);
4   }
```

- Vereinigung identischer Funktionen und Konstanten (O2, Os)
- Umordnen des Codes (O, O2, O3, Os)
  - Erhöhung der Lokalität
  - Reduzierung von Sprüngen in Verzweigungen
  - Verzweigungen vor Schleifen

Weitere Optimierungsmethoden

- Kleinere Datentypen
  - z.B.: uint8_t für positive Werte bis 255
- Multidimensionale Arrays
  - Iteration über den letzten Index zuerst
- Pointer an Stelle von Daten übergeben
  - Nur auf dem Heap möglich
- Keine ungenutzten return-Werte übergeben

Zusammenfassung

- Statische Speicherverwaltung: Stack
  - Einschränkungen
- Dynamische Speicherverwaltung: Heap
  - Nutzen und Risiken
- Speicheroptimierung
  - Data Alignment
  - Unions
  - Compiler

Literatur 1

- Speicherverwaltung
  - Memory: Stack vs Heap, Paul Gribble, Summer 2012:
    - https://www.gribblelab.org/CBootCamp/7_Memory_Stack_vs_Heap.html
  - Programming in C, Bharat Kinariwala & Tep Dobry:
    - http://www-ee.eng.hawaii.edu/~tep/EE160/Book/
  - CS 5641, Rich Maclin, University of Minnesotah Duluth, Fall 2009:
    - https://www.d.umn.edu/~rmaclin/cs5641/Notes/L18_MemoryManagement.pdf

- Cache Management
  - Beyond Physical Memory: Policies, Remzi H. Arpaci-Dusseau & Andrea C. Arpaci-Dusseau, Arpaci-Dusseau Books, 2014
    - http://pages.cs.wisc.edu/~remzi/OSTEP/vm-beyondphys-policy.pdf

Literatur 2

- Buffer Overflow
  - Buffer Overflow Attack, Thomas Schwarz, Santa Clara University, 2004:
    - http://www.cse.scu.edu/~tschwarz/coen152_05/Lectures/BufferOverflow.html

- Data Alignment
  - Data Alignment, Song Ho Ahn, 2011:
    - http://www.songho.ca/misc/alignment/dataalign.html

- Optimierungsmethoden:
  - Embedded C – Optimization techniques, Emertxe Information Technologies Pvt Ltd, 2014-3-25:
    - https://de.slideshare.net/EmertxeSlides/embedded-c-optimization-techniques

Literatur 3

- Dynamische Speicherverwaltung in C: Library Functions
  - https://www.tutorialspoint.com/c_standard_library/c_function_malloc.htm
  - https://www.tutorialspoint.com/c_standard_library/c_function_calloc.htm
  - https://www.tutorialspoint.com/c_standard_library/c_function_realloc.htm
  - https://www.tutorialspoint.com/c_standard_library/c_function_free.htm

- GCC Optimierungsoptionen
  - https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html