

Verlässliche Echtzeitsysteme

Übungen zur Vorlesung

Phillip Raffeck, Florian Schmaus, Simon Schuster

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)
<https://www4.cs.fau.de>

Sommersemester 2019



- 1 C-Quiz Teil VI
- 2 Stack- & Laufzeitanalyse
- 3 Aufgabenstellung



1 C-Quiz Teil VI

2 Stack- & Laufzeitanalyse

3 Aufgabenstellung



- C99
- x86 bzw. x86-64, d. h.
 - vorzeichenbehaftete Integer als Zweierkomplement implementiert
 - char hat 8 Bit
 - short hat 16 Bit
 - int hat 32 Bit
 - long hat 32 Bit auf x86 und 64 Bit auf x86-64



Frage 17

Angenommen x hat Typ `int`. Ist $x - 1 + 1 \dots$

1. definiert für alle Werte
 2. definiert für manche Werte
 3. definiert für keinen Wert
- von x ?



Frage 17

Angenommen x hat Typ `int`. Ist $x - 1 + 1 \dots$

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- additive Operatoren sind linksassoziativ
- ⇒ nicht definiert für `INT_MIN`



Frage 18

Angenommen x hat Typ `int`. Ist `(short)x + 1 ...`

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?



Frage 18

Angenommen x hat Typ `int`. Ist `(short)x + 1 ...`

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- wenn x nicht in `short` passt

↪ Verhalten ist implementierungsabhängig



Frage 19

Angenommen x hat Typ `int`. Ist `(short) (x + 1) ...`

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?



Frage 19

Angenommen x hat Typ `int`. Ist `(short) (x + 1) ...`

1. definiert für alle Werte
2. definiert für manche Werte
3. definiert für keinen Wert

von x ?

Erklärung

- wenn $x + 1$ nicht in `short` passt
 - ↳ implementierungsabhängig
- die meisten Compiler schneiden beim Cast ab



Frage 20

Hat die Auswertung von `INT_MIN % -1` definiertes Verhalten?

1. ja
2. nein



Frage 20

Hat die Auswertung von `INT_MIN % -1` definiertes Verhalten?

1. ja
2. nein
3. das weiß niemand ...

Erklärung

- C99 macht dazu keine Aussage
- in C11 gilt folgendes:
 - wenn $(a/b)*b + a\%b$ darstellbar ist, haben a/b und $a\%b$ definiertes Verhalten
 - sonst nicht
 - $\text{INT_MIN} / -1$ entspricht $\text{INT_MAX} + 1$
 - was auf x86/x86-64 nicht darstellbar ist



1 C-Quiz Teil VI

2 Stack- & Laufzeitanalyse

3 Aufgabenstellung



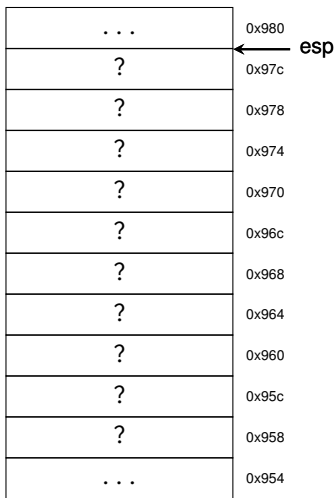


- Harte, verlässliche Echtzeitsysteme
 - Garantien über Ressourcenbedarf notwendig
 - ☞ statische Analyse unabdingbar
- Mögliche Ressourcen: Speicherbedarf, Laufzeit, etc.
- Übung: Analyse des Stackverbrauchs einer Feldbibliothek
- Stack-Analyse
 1. Dynamisch: Wasserstandstechnik
 2. Statisch: „Eigenbau“ und aiT (Stack-Analyzer der a³ Suite)
- WCET-Analyse mittels aiT (bereits in EZS behandelt)

Beispiel: Programmstapel

```
int main(void) {  
→ return f(4, 2);  
}
```

```
52a: push ebp  
52b: mov ebp, esp  
52d: lea ...  
534: or ...  
538: lea ...  
→ 53f: push 0x2  
541: push 0x4  
543: call 4fd <f>  
548: add esp, 0x8  
54b: leave  
54c: ret
```



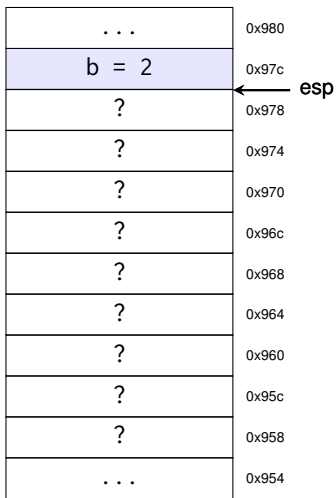
Wachstumsrichtung



Beispiel: Programmstapel

```
int main(void) {  
→ return f(4, 2);  
}
```

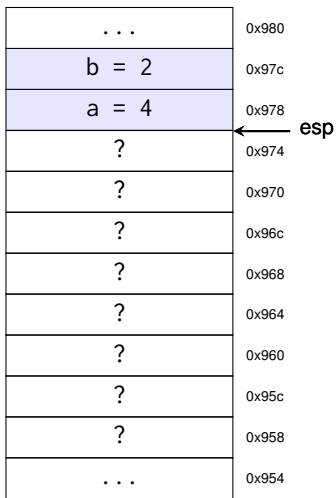
```
52a: push ebp  
52b: mov ebp, esp  
52d: lea ...  
534: or ...  
538: lea ...  
53f: push 0x2  
→ 541: push 0x4  
543: call 4fd <f>  
548: add esp, 0x8  
54b: leave  
54c: ret
```



Beispiel: Programmstapel

```
int main(void) {  
→   return f(4, 2);  
}
```

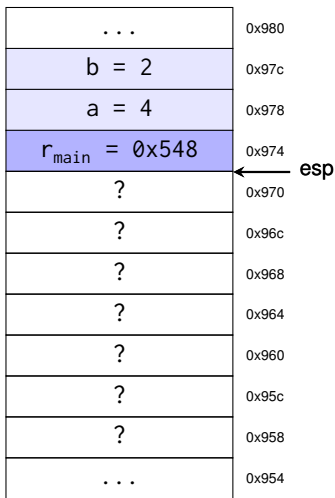
```
52a:  push ebp  
52b:  mov  ebp,esp  
52d:  lea  ...  
534:  or   ...  
538:  lea  ...  
53f:  push 0x2  
541:  push 0x4  
→ 543:  call 4fd <f>  
548:  add  esp,0x8  
54b:  leave  
54c:  ret
```



Beispiel: Programmstapel

```
→ int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

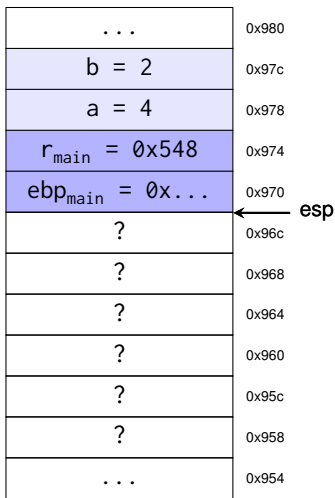
```
→ 4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
→ int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

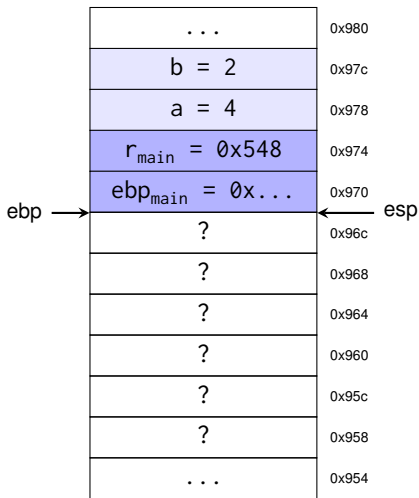
```
4fd:  push ebp  
→ 4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
→ int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

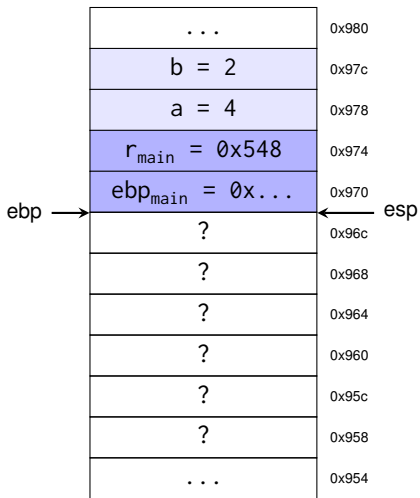
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
→ 500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
→ int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

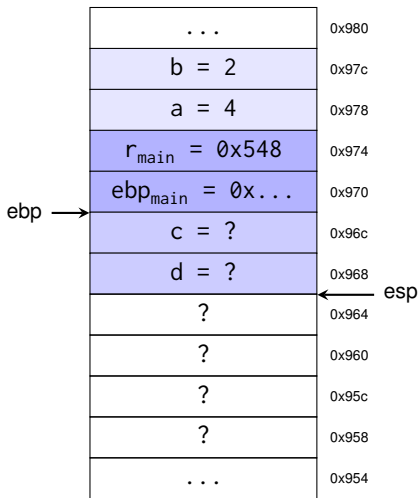
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
→ 50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

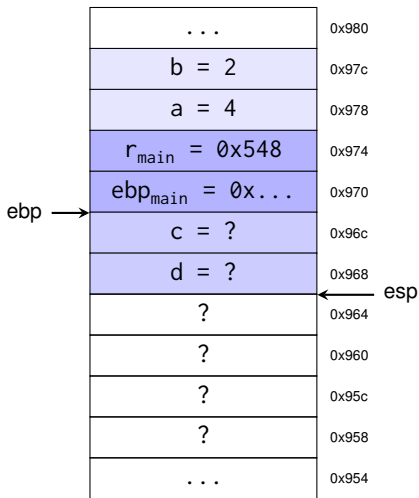
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
→ int c = a + b;  
  int d = g(c);  
  return d;  
}
```

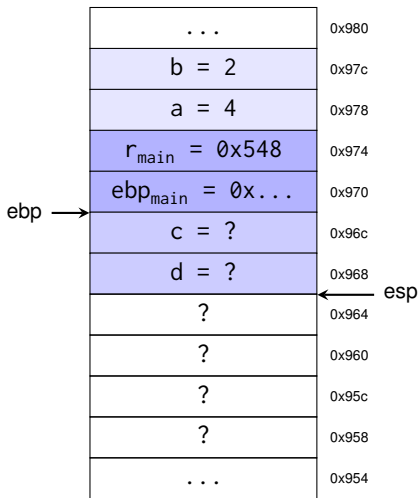
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
→ 515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
→ int c = a + b;  
  int d = g(c);  
  return d;  
}
```

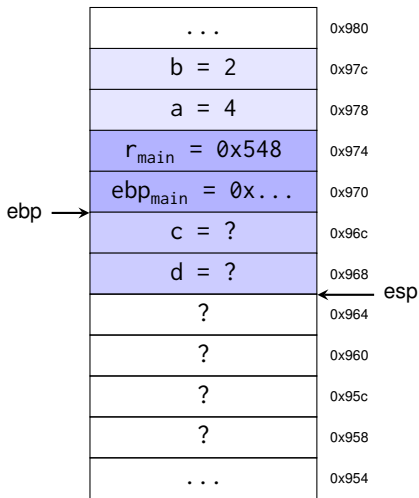
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
→ 518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
→ int c = a + b;  
  int d = g(c);  
  return d;  
}
```

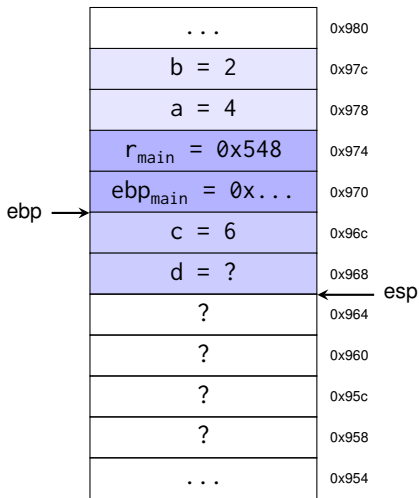
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
→ 51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

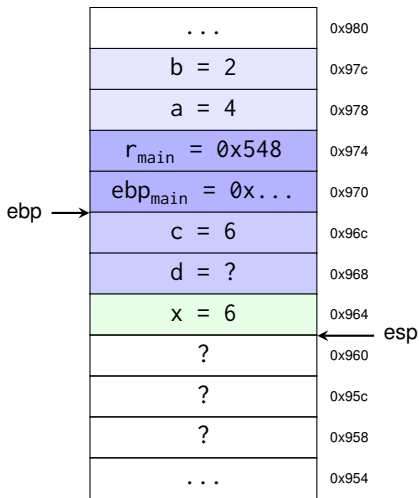
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

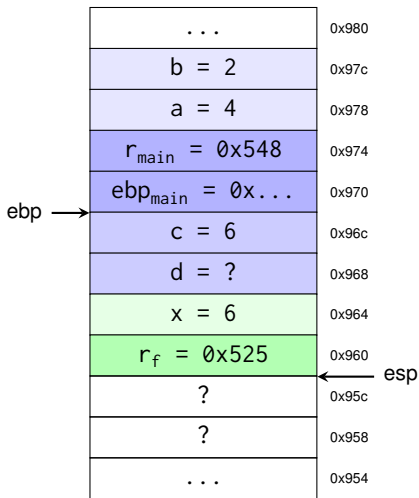
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
→ int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

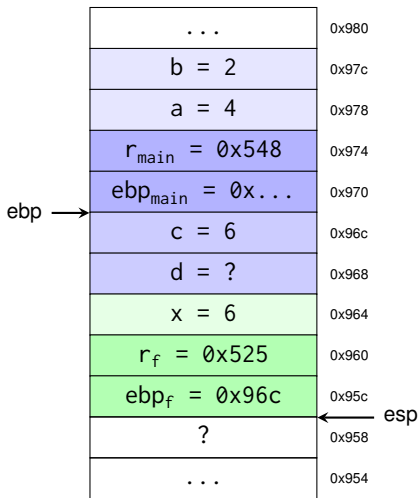
```
→ 4e9:  push ebp  
4ea:  mov  ebp,esp  
4ec:  sub  esp,0x4  
4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
4f5:  mov  DWORD PTR [ebp-0x4],eax  
4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
→ int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

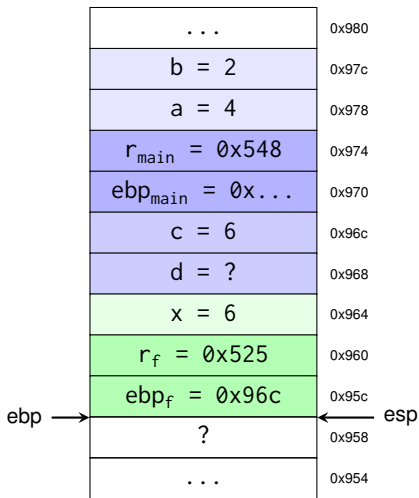
```
→ 4e9:  push ebp  
4ea:  mov  ebp,esp  
4ec:  sub  esp,0x4  
4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
4f5:  mov  DWORD PTR [ebp-0x4],eax  
4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
→ int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

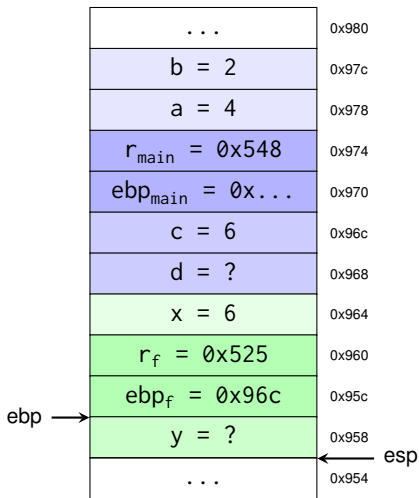
```
4e9:  push ebp  
4ea:  mov  ebp,esp  
→ 4ec:  sub  esp,0x4  
4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
4f5:  mov  DWORD PTR [ebp-0x4],eax  
4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
→ int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

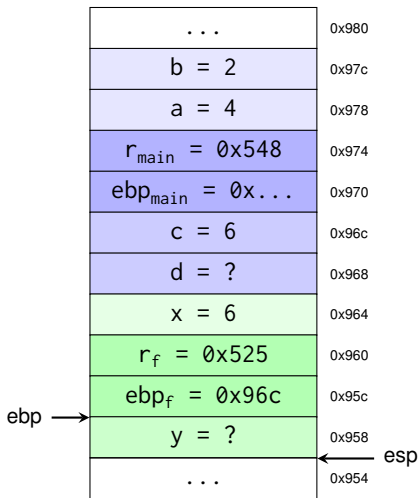
```
4e9:  push ebp  
4ea:  mov  ebp,esp  
4ec:  sub  esp,0x4  
→ 4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
4f5:  mov  DWORD PTR [ebp-0x4],eax  
4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
→ int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

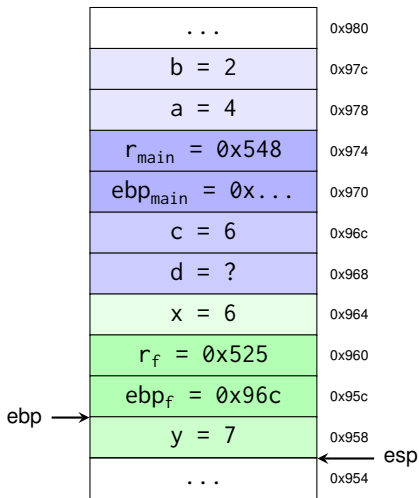
```
4e9:  push ebp  
4ea:  mov  ebp,esp  
4ec:  sub  esp,0x4  
→ 4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
4f5:  mov  DWORD PTR [ebp-0x4],eax  
4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
→ int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

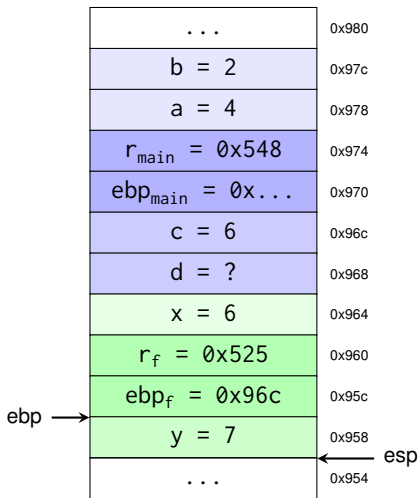
```
4e9:  push ebp  
4ea:  mov  ebp,esp  
4ec:  sub  esp,0x4  
4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
→ 4f5:  mov  DWORD PTR [ebp-0x4],eax  
4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
→ int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

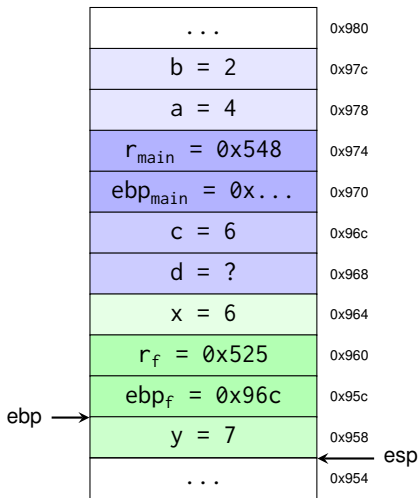
```
4e9:  push ebp  
4ea:  mov  ebp,esp  
4ec:  sub  esp,0x4  
4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
4f5:  mov  DWORD PTR [ebp-0x4],eax  
→ 4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

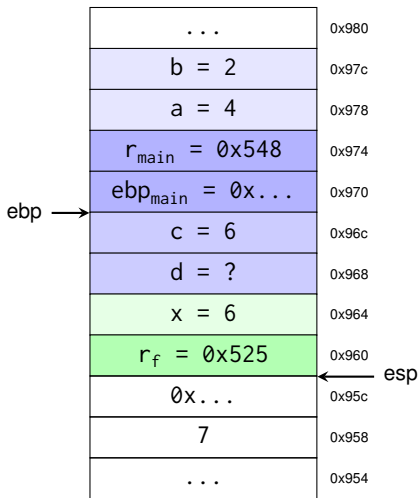
```
4e9:  push ebp  
4ea:  mov  ebp,esp  
4ec:  sub  esp,0x4  
4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
4f5:  mov  DWORD PTR [ebp-0x4],eax  
4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
int g(int x) {  
    int y = x + 1;  
    return y;  
}
```

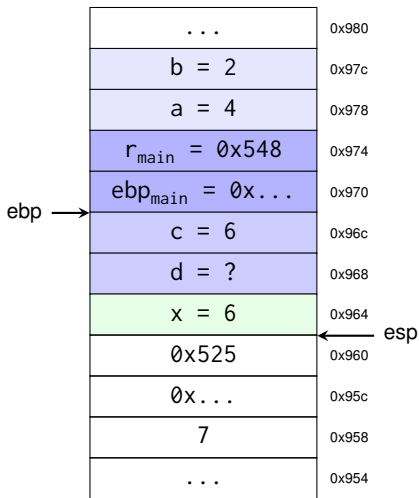
```
4e9:  push ebp  
4ea:  mov  ebp,esp  
4ec:  sub  esp,0x4  
4ef:  mov  eax,DWORD PTR [ebp+0x8]  
4f2:  add  eax,0x1  
4f5:  mov  DWORD PTR [ebp-0x4],eax  
4f8:  mov  eax,DWORD PTR [ebp-0x4]  
4fb:  leave  
4fc:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

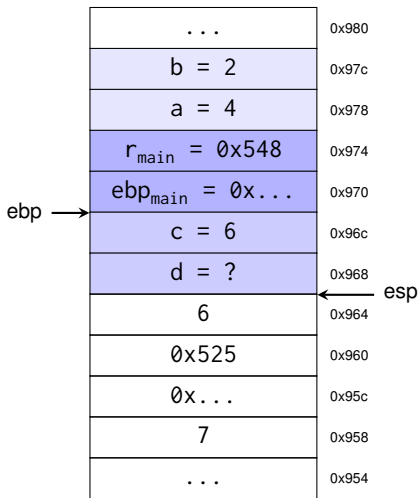
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

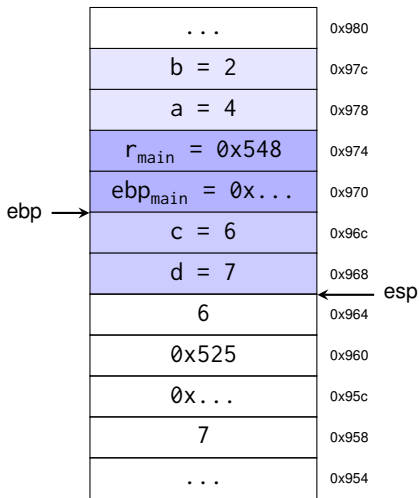
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

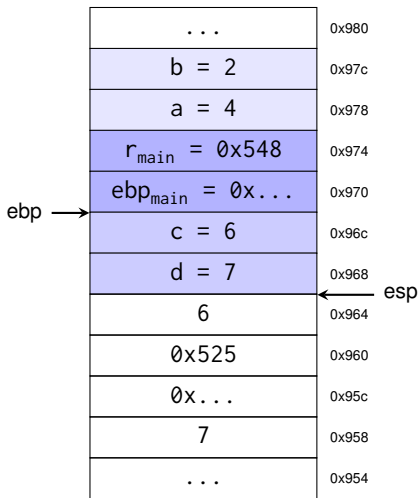
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

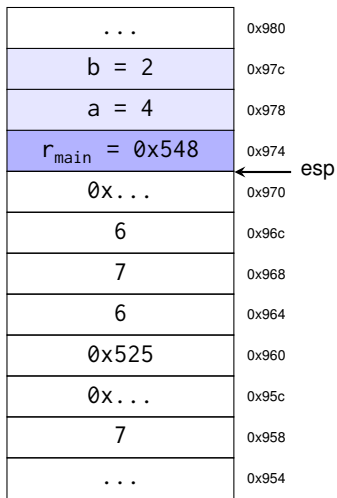
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
int f(int a, int b) {  
    int c = a + b;  
    int d = g(c);  
    return d;  
}
```

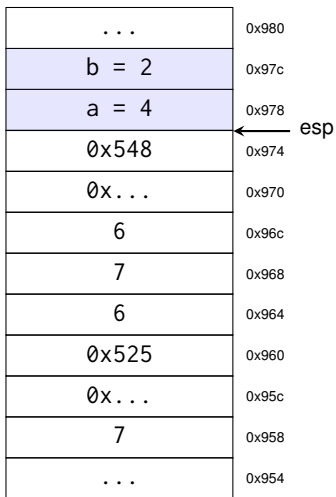
```
4fd:  push ebp  
4fe:  mov  ebp,esp  
500:  lea  ...  
507:  or   ...  
50b:  sub  esp,0x8  
512:  mov  edx,DWORD PTR [ebp+0x8]  
515:  mov  eax,DWORD PTR [ebp+0xc]  
518:  add  eax,edx  
51a:  mov  DWORD PTR [ebp-0x8],eax  
51d:  push DWORD PTR [ebp-0x8]  
520:  call 4e9 <g>  
525:  add  esp,0x4  
528:  mov  DWORD PTR [ebp-0x4],eax  
52b:  mov  eax,DWORD PTR [ebp-0x4]  
52e:  leave  
52f:  ret
```



Beispiel: Programmstapel

```
→ int main(void) {  
    return f(4, 2);  
}
```

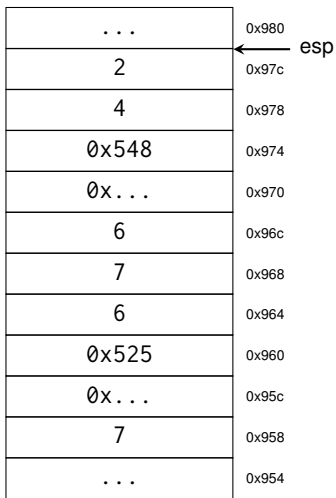
```
52a: push ebp  
52b: mov ebp, esp  
52d: lea ...  
534: or ...  
538: lea ...  
53f: push 0x2  
541: push 0x4  
→ 543: call 4fd <f>  
548: add esp, 0x8  
54b: leave  
54c: ret
```



Beispiel: Programmstapel

```
→ int main(void) {  
    return f(4, 2);  
}
```

```
52a:  push ebp  
52b:  mov  ebp,esp  
52d:  lea  ...  
534:  or   ...  
538:  lea  ...  
53f:  push 0x2  
541:  push 0x4  
543:  call 4fd <f>  
548:  add  esp,0x8  
→ 54b:  leave  
54c:  ret
```



Beispiel: Programmstapel

```
int main(void) {  
    return f(4, 2);  
}
```

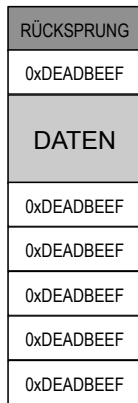
```
52a:  push ebp  
52b:  mov  ebp,esp  
52d:  lea  ...  
534:  or   ...  
538:  lea  ...  
53f:  push 0x2  
541:  push 0x4  
543:  call 4fd <f>  
548:  add  esp,0x8  
54b:  leave  
54c:  ret
```

...	0x980
2	0x97c
4	0x978
0x548	0x974
0x...	0x970
6	0x96c
7	0x968
6	0x964
0x525	0x960
0x...	0x95c
7	0x958
...	0x954

Wachstumsrichtung



- **Messung zur Laufzeit:** Water-Marking (siehe Vorlesung)
- Grundidee: Einfügen von **Stack Canaries**
- Explizite Verwaltung des Stapelspeichers notwendig
- pthread-Bibliothek ermöglicht Verwaltung
- Mögliche Canaries
 - Lesbare Bitmuster: 0xDEADBEEF
 - Unwahrscheinliche Bitmuster: 0b101010101010...





1. (Globalen) Stack anlegen:

```
1 static unsigned int g_data[DATA_SIZE];
```

2. Thread anlegen & starten:

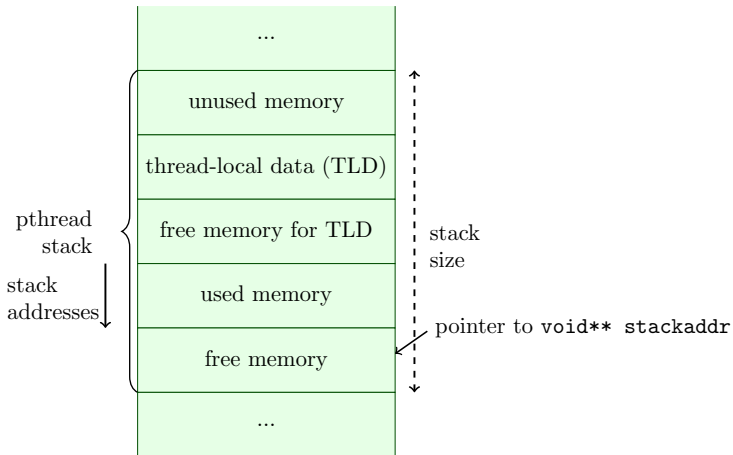
```
1 pthread_t thread;  
2 pthread_attr_t attr;  
3 pthread_attr_init(&attr);  
4 pthread_attr_setstack(&attr, &g_stack, STACK_SIZE);  
5 // worker function: void *run(void *param)  
6 int status = pthread_create(&thread, &attr, run, NULL);  
7 if (status != 0) { ... // handle error
```

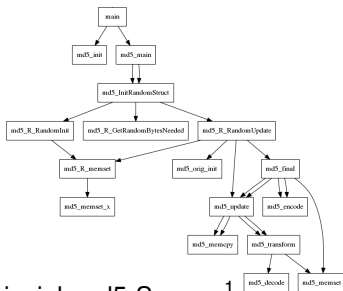
3. Auf Thread warten:

```
1 pthread_join(thread, &ret);
```



pthread Stack





```

1  /* Objective function */
2  max: +16 md5_orig_init +64 md5_updat
3      +16 md5_memset +208 md5_transform
4
5
6  /* Constraints */
7  +main = 1;
8  +md5_init +md5_main <= +main;
9  ...
    
```

■ Beispiel: md5-Summe¹

■ Vorgehen

1. Callgraph bestimmen
2. Stackverbrauch einzelner Funktionen (gcc -fstack-usage)
3. ILP² aufstellen (Nebenbedingungen aus 1., Kosten aus 2. verwenden)
4. ILP z.B. mittels lp_solve \leadsto **worst-case Stackverbrauch**

¹<https://github.com/tacle/tacle-bench/>

²Integer Linear Program (dt. ganzzahliges lineares Programm)

Optimierungsziel

- Jeder Stapelrahmen einer Funktion f hat eine Größe $size$
- Jede Funktion kann auf einem Pfad ein- oder mehrfach (Rekursion), insgesamt n -fach auf dem Stapel vorkommen
- Gesucht ist Pfad durch den Aufrufgraphen, welcher Stapelbedarf maximiert
- Dabei müssen **Flussbedingungen** eingehalten werden
 - Aufruferbeziehung
 - Alternativen
 - ...

Optimierungsziel

$$\max \sum_{\text{Funktion } f} size_f \cdot n_f$$

In lp_solve -Syntax: `max : +64 n_f1 +48 n_f2 +42 n_f3 ;`



Flussbedingung: Initialer Aufruf

Semantik

Der initiale Aufruf erfolgt maximal (wahlweise auch genau) ein mal

Formalisierung

$$n_{\text{main}} \leq 1$$



lp_solve -Syntax

```
n_main <= 1;
```



Flussbedingung: Mehrere Vorgänger

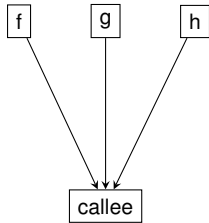
Semantik

Jede Funktion kann nur so oft ausgeführt werden, wie sie von den Vorgängern aus aufgerufen wird

Formalisierung

Sei $f_{a \rightarrow b}$ die Anzahl der Aufrufe von b durch a:

$$n_{callee} \leq \sum_{p \in \text{Aufrufer}(callee)} f_{p \rightarrow callee}$$



lp_solve -Syntax

```
n_caller <= + f_f_callee + f_g_callee + f_h_callee ;
```



Flussbedingung: Immer nur ein Nachfolger pro Funktion

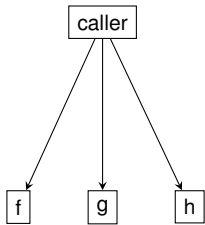
Semantik

Jede Funktionsinkarnation ruft gleichzeitig jeweils maximal eine weitere Funktion auf

Formalisierung

Sei $f_{a \rightarrow b}$ die Anzahl der Aufrufe von b durch a :

$$\sum_{c \in \text{Aufgerufene}(\text{caller})} f_{\text{caller} \rightarrow c} \leq n_{\text{caller}}$$



lp_solve -Syntax

```
+ f_caller_f + f_caller_g + f_caller_h <= n_caller ;
```



Flussbedingung: Rekursion

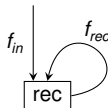
Semantik

Rekursive Funktionen können pro Aufruf von außen bis zu ihrer maximalen Rekursionstiefe (d) oft ausgeführt werden.

Formalisierung

$$f_{rec} \leq d_{rec} \cdot f_{in}$$

$$n_{rec} \leq f_{in} + f_{rec}$$



lp_solve -Syntax

```
f_rec <= +42 f_in ;  
n_rec <= f_in + f_rec ;
```



- Problemformulierung in Ipsolve:

```
max: +40 n_main +20 n_f +60 n_g;
```

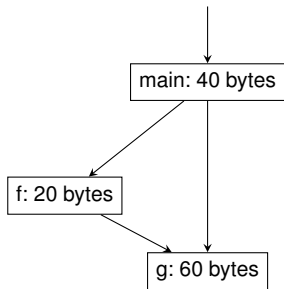
```
n_main <= 1;
```

```
+f_main_f +f_main_g <= n_main;
```

```
n_f <= +f_main_f;
```

```
+f_f_g <= n_f;
```

```
n_g <= +f_f_g +f_main_g;
```



Beispiel

- Problemformulierung in lpsolve:

```
max: +40 n_main +20 n_f +60 n_g;
```

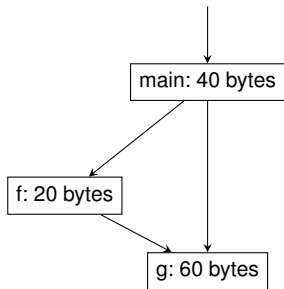
```
n_main <= 1;  
+f_main_f +f_main_g <= n_main;  
n_f <= +f_main_f;  
+f_f_g <= n_f;  
n_g <= +f_f_g +f_main_g;
```

- Ausgabe von lp_solve :

```
Value of objective function: 120.00000000
```

```
Actual values of the variables:
```

```
n_main          1  
n_f             1  
n_g             1  
f_main_f       1  
f_main_g       0  
f_f_g          1
```



```
$ lp_solve infeasible.lp  
This problem is infeasible
```

Infeasible Models

Eine oder mehrere der Nebenbedingungen führen zu einem logischen Widerspruch

Leider bietet `lp_solve` selbst direkt keine Hilfestellung zur Lokalisation. Die Entwickler empfehlen das Einführen von “slack”-Variablen:³

<code>max: x + y;</code>	<code>max: x + y;</code>	<code>x: 20</code>
<code>x + 1 <= x;</code>	<code>x + 1 - e_1 <= x;</code>	<code>y: 20</code>
<code>y > y + 1;</code>	<code>y + e_2 > y + 1;</code>	<code>e_1: 1</code>
<code>x <= 20;</code>	<code>x <= 20;</code>	<code>e_2: 1</code>
<code>y <= 20;</code>	<code>y <= 20;</code>	

³<http://lpsolve.sourceforge.net/5.5/Infeasible.htm>

```
$ lp_solve unbounded.lp  
This problem is unbounded
```

Unbounded Models

Eine oder mehrere der Variablen sind nach oben unbeschränkt

Durch künstliche Beschränkung aller Variablen im System (auf einen sehr großen Wert) lassen sich unbeschränkte Variablen detektieren:

max: $x + y + z$;	max: $x + y + z$;	x: 5000
$z \leq y + 1$;	$z \leq y + 1$;	y: 20
$y \leq 20$;	$y \leq 20$;	z: 21
	$x \leq 5000$;	
	$y \leq 5000$;	
	$z \leq 5000$;	



- `lp_solve` ist auf die Lösung linearer Gleichungssysteme ausgelegt
- Es ist dementsprechend nicht möglich, zwei Variablen zu multiplizieren
 - `a * b` \Rightarrow Syntaxfehler
 - `max : a b` \Rightarrow optimiert $a + b$
- Lösung in VEZS für Konstanten (Stapelrahmengrößen): C-Präprozessor:

```
#define s_main 40  
#define s_f    20  
#define s_g    60
```

```
max: +s_main n_main +s_f n_f +s_g n_g;
```

~> `stackusage / lp_solvepp`



- Statische Code-Analyse mit a³ Tool-Suite
 1. aiT: WCET-Analyse
 2. Stack-Analyzer: Stackverbrauch
 3. ...
- Installiert im CIP-Pool
- Verfügbare Rechner:
 - CIP2: faui00a-y
 - CIP2: faui02a-y
 - CIP2: faui0fa-u



a³ Analyzer – Neue Projekt Anlegen

The screenshot shows the 'New' menu of the a3 Analyzer. The menu items are:

- Open... (Ctrl+O)
- Recently opened
- Save project
- Save as project...
- Save as workspace...
- Import project
- Compare projects...
- Start all enabled analyses (F6)
- Start analysis (F7)
- Start analysis interactive (F8)
- Stop analysis (F4)
- Preferences... (Ctrl+,)
- Quit (Ctrl+Q)

The main window displays a list of analysis tools:

- ValueAnalyzer**: Program value analysis
- ResultCombinator**: Combination of results according to formula
- Control-Flow Visualizer**: Visualization of control-flow graph

At the bottom, there are search and memory usage indicators.



a³ Analyzer – Executable Angeben

The screenshot shows the a³ Analyzer interface. On the left is a sidebar with a tree view containing categories like Home, Analyses, Setup, and Information, with sub-items such as Files, Hardware, Reporting, Decoding, Stack & value analysis, Timing analysis, Source files, Configurations, and Support. The main window has a menu bar (Project, Views, Help) and a title bar (Files). The 'Executables:' section features a large red rectangular area with a 'Browse for item' button. Below this are three input fields: 'AIS file:', 'Report file:', and 'XML results file:'. At the bottom, the 'Project:' section shows the 'Temporary directory: /tmp/a3arm-psjBMK' and buttons for 'Search' and 'Memory usage'.



a³ Analyzer – Annotations-Datei Anlegen

Project Views Help

Files

- stackanalyzer_arm.ais
- Hardware
- Reporting
- Decoding
- Stack & value analysis
- Timing analysis
- Source files
- Configurations
- Support

Home

Analyses

f(x)

Setup

Information

Files

Executables: `lehre/vezs-aufgaben/06_Analyse/build-arm/stackanalyzer_arm`

AIS file: `stackanalyzer_arm.ais` [Edit AIS file](#)

Report file:

XML results file:

Project:

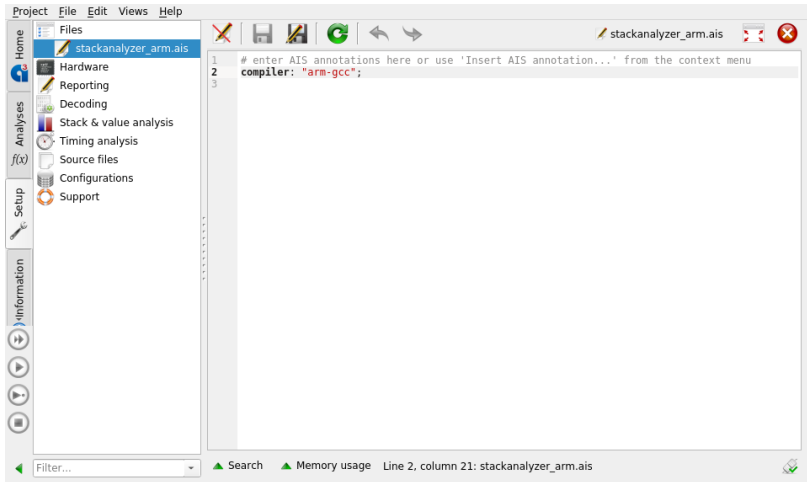
Temporary directory: `/tmp/a3arm-psjBMK`

Filter...

Search Memory usage



a³ Analyzer – Compiler-Annotation Anlegen



The screenshot displays the a3 Analyzer IDE interface. The top menu bar includes 'Project', 'File', 'Edit', 'Views', and 'Help'. The toolbar contains icons for file operations (delete, save, copy, paste, undo, redo) and window management (maximize, close). The left sidebar shows a project tree with categories: Home (Files, stackanalyzer_arm.ais), Hardware, Reporting, Decoding, Analyses (Stack & value analysis, Timing analysis), f(x) (Source files, Configurations), Setup (Support), and Information. The main editor window shows the file 'stackanalyzer_arm.ais' with the following content:

```
1 # enter AIS annotations here or use 'Insert AIS annotation...' from the context menu
2 compiler: "arm-gcc";
3
```

The status bar at the bottom indicates 'Line 2, column 21: stackanalyzer_arm.ais' and includes search and memory usage icons.



a³ Analyzer – Stack-Analyse Selektieren

The screenshot shows the a³ Analyzer software interface. At the top, there is a menu bar with 'Project', 'Views', and 'Help'. Below the menu bar is a toolbar with a 'f(x) Create' button. The main window is titled 'f(x)' and contains a list of analysis tools. The tools are:

- aiT**: Safe WCET analysis (represented by a clock icon)
- StackAnalyzer**: Stack usage analysis (represented by a bar chart icon)
- TimeWeaver**: Trace-based WCET estimation (represented by a clock and trace icon)
- ValueAnalyzer**: Program value analysis (represented by a binary code icon)
- ResultCombinator**: Combination of results according to formula (represented by a chalkboard icon)
- Control-Flow Visualizer**: Visualization of control-flow graph (represented by a flowchart icon)

At the bottom of the main window, there is a 'Filter...' dropdown menu and two buttons: 'Search' and 'Memory usage'. The left sidebar contains a vertical menu with 'Home', 'Analyses', 'Setup', and 'Information' buttons.



a³ Analyzer – Stack-Analyse Starten

The screenshot displays the a3 Analyzer interface. At the top, there is a menu bar with 'Project', 'Analysis', 'Views', and 'Help'. Below the menu bar is a toolbar with icons for file operations and analysis. The main window is titled 'StackAnalyzer' and contains the following fields:

- ID: StackAnalyzer
- Comment: (empty)
- Result: n/a

Below these fields are two tabs: 'Settings' and 'Output'. The 'Settings' tab is active and contains the following configuration options:

- Configuration: Default Configuration
- Dependencies: (empty)
- Analysis start: main
- AIS file: (empty)
- Expected result: (empty)

At the bottom of the window, there is a status bar with a 'Filter...' dropdown and three indicators: Messages, Search, and Memory usage.



a³ Analyzer – Stack-Analyse Starten

The screenshot shows the StackAnalyzer application window. The title bar includes the menu items "Project", "Analysis", "Views", and "Help". The main window is divided into a left sidebar and a main content area.

Left Sidebar:

- Buttons: Home, f(x) Create, StackAnalyzer, f(x), Setup, Information.
- Navigation icons: Home, Back, Forward, Stop, Refresh, Close.
- Item: "Start all enabled analyses" (highlighted in yellow).
- Filter... dropdown at the bottom.

Main Content Area:

- Toolbar: Home, .ais, Run.
- Window title: StackAnalyzer.
- Form fields:
 - ID: StackAnalyzer
 - Comment: (empty)
 - Result: n/a
- Settings tab (selected) and Output tab.
- Configuration: Default Configuration (dropdown)
- Dependencies: (dropdown)
- Analysis start: main (text input)
- AIS file: (text input with file icon)
- Expected result: (text input)

At the bottom of the main area, there are status indicators for Messages, Search, and Memory usage, along with a green checkmark icon.



1 C-Quiz Teil VI

2 Stack- & Laufzeitanalyse

3 Aufgabenstellung



- Existierende Implementierung: Array-Datenstruktur
- Vorgegebene Funktionen: Sortieren, Maximumssuche, ...
- Aufgaben
 1. Dynamische Analyse
 - 1.1 Thread erstellen
 - 1.2 Stack initialisieren
 - 1.3 Programm (mit Eingabedaten) ausführen
 - 1.4 Stackverbrauch messen
 2. Statische Analyse
 - 2.1 ILP aus Aufrufgraph aufstellen
 - 2.2 Mittels `lp_solve` lösen
 - 2.3 Optional: Verwendung `a3` Stack-Analyzer



Fragen?

