Mixed Language Programming

PROSEMINAR EFFIZIENTE PROGRAMMIERUNG SOSE 2018

POLINA RAJKO

Einführung

- Java

```
public class Main
{
    public static int quadrat(int i) {
        return i*i;
}
    public static void main(String[] args) {
       for ( i = 1; i<=10; i++ ) {
            System.out.println( i + “ “ + quadrat(i));
        }
     }
}
```

Einführung

- C

```
#include<stdio.h>

int quadrat(int i) {
        return i*i;
}

int main(void) {
        int i;
    for (i = 1; i<=10; i++) {
        printf( %d %d \ n, i, quadrat(i)) }
    return 0;
}
```

Einführung

- Haskell

```
quadrat x = x*x

main = putstr $ concatMap ( \i -> show i ++ “  “ ++ show ( quadrat i ) ++ “ \n“ ) [1..10]
```

- Python

```
def quadrat (i):
        return (i*i)

for i in range(1,11):
        print( i , quadrat(i))
```

Unterschiede

Im Bezug auf:

- Programmierparadigma (prozedural, objektorientiert, funktional)
- Performance und Speicherverwendung
- (intuitive) Verwendbarkeit, und die benötigte Zeit zum Lernen
- oft sehr spezielle Aufgaben

Viele Arten der Implementation einer Funktion, jedoch keine direkte Codeübernahme möglich

Mixed Language Programming

- Programmieren mit mehr als einer Programmiersprache
- Kombination von Programmiersprachen
- Wiederverwendung von bereits geschriebenem Code möglich
- Einbinden von Modulen anderer Programmiersprachen
- Eignung für große Projekte → Informationsymmetrie nicht benötigt
- Hypothetisch: Gute Basis zur Entwicklung neuer Programmiersprachen → D

D

- Entwickelt durch Andrej Alexandrescu, Walter Bright (2001/2007)
- Schwerpunktlegung auf Performance, Mächtigkeit, schnelle Erlernbarkeit
- Orientierung an C++, Python, Ruby und PHP
- Nutzer: Facebook, Netflix, Ebay
-

```
import std.stdio;

void main()
{
    writeln(“Hello!“);
}
```

Mögliche Problemstellungen

- Fundamentale Unterschiede zwischen Programmiersprachen
- Abweichende Implementationsweisen
- Verlust von Übersichtlichtkeit
- Probleme zwischen Interfaces (fehlerhafte/undurchdachte Implementationen)

Mögliche Problemstellungen

- Benötigtes Hintergrundwissen
- Einschränkung der Benutzerumgebungen
- Große Software- und Compilernutzung benötigen Wartung
- „fehlende Ästhetik“

Wie programmiert man mit mehr als einer Programmiersprache?

- Grundsätzliche Vorraussetzung: Aufteilung des Systems in Unterprogramme,

  Herstellung von Interface – Benutzer – Verhätnissen zwischen diesen

- Unterschiede in den Vorgehensweisen durch ihre Verwendung:
  1. eines automatischen Systems
  2. einer Virtual Program Library
  3. einer Bindung von Subroutine Libraries
  4. von Datentypen einer der Programmiersprachen
  5. einer Kombinierung bereits vollständiger Programme
  6. von Tools

1. Durch die Verwendung eines automatischen Systems

- System (Compiler, Linker, Loader) für die Kompilierung, Interpretation und Zusammensetzen der entstehenden Unterprogramme verantwortlich
- Gleichbehandlung aller Sprachen
- Vorteile: - Übernahme aller Aufgaben durch das System,
  Nutzer nur für die Verwendung von erlaubten Sprachen verantwortlich
- Nachteile: - Einschränkungen durch die Vorgabe bestimmter Sprachkombinationen

1. Durch die Verwendung eines automatischen Systems

- Grundidee:

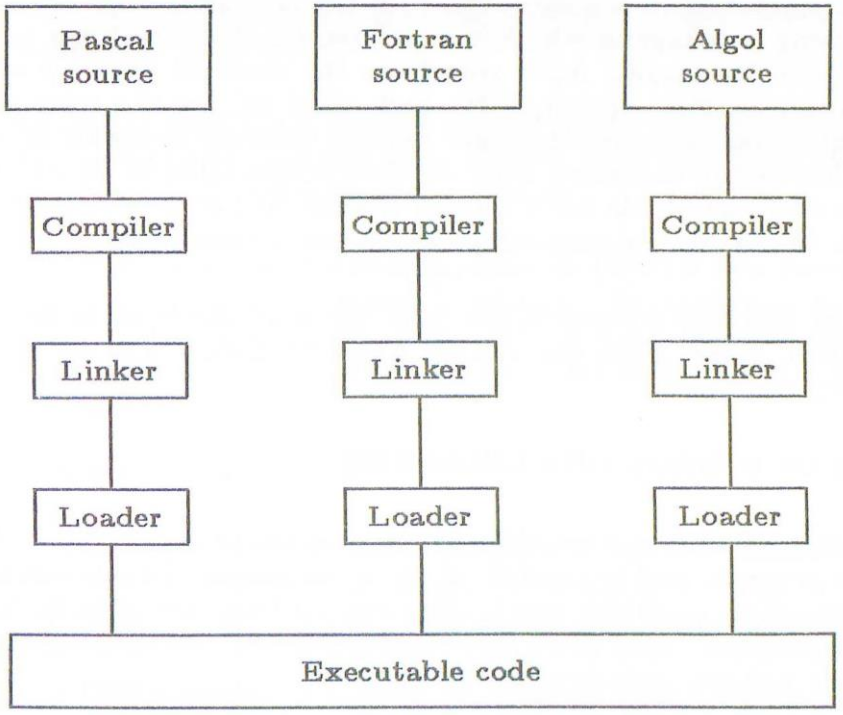


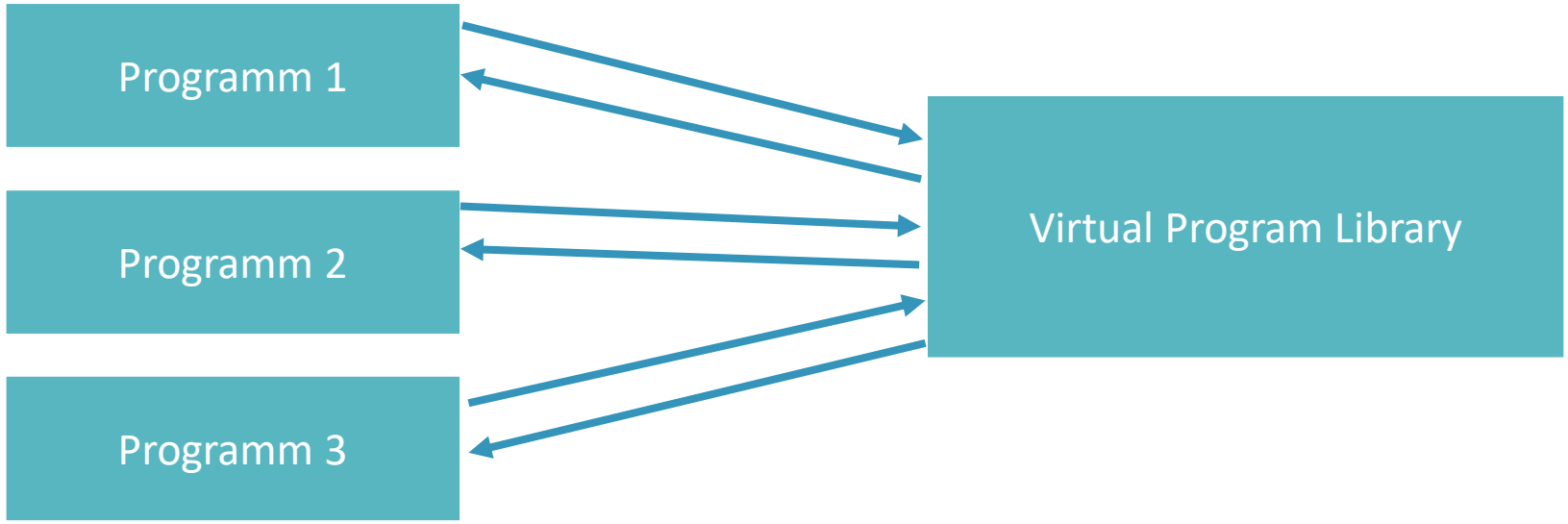
Abb. 1

2. Durch die Verwendung einer Virtual Program Library

- Virtual Program Library: Library, in einer computernahen, für verschiedene Programmierumgebungen geeigneten, Programmiersprache
- Möglichkeit des Aufrufs dieser über nutzernahe Programmiersprachen
- Vorteile: - Aufrufe durch verschiedene Sprachen möglich
  - Wiederverwendung von Code
- Nachteile: - genaue, spezifische Angaben des Programmierers zur Nutzung benötigt

2. Durch die Verwendung einer Virtual Program Library

- Grundidee:

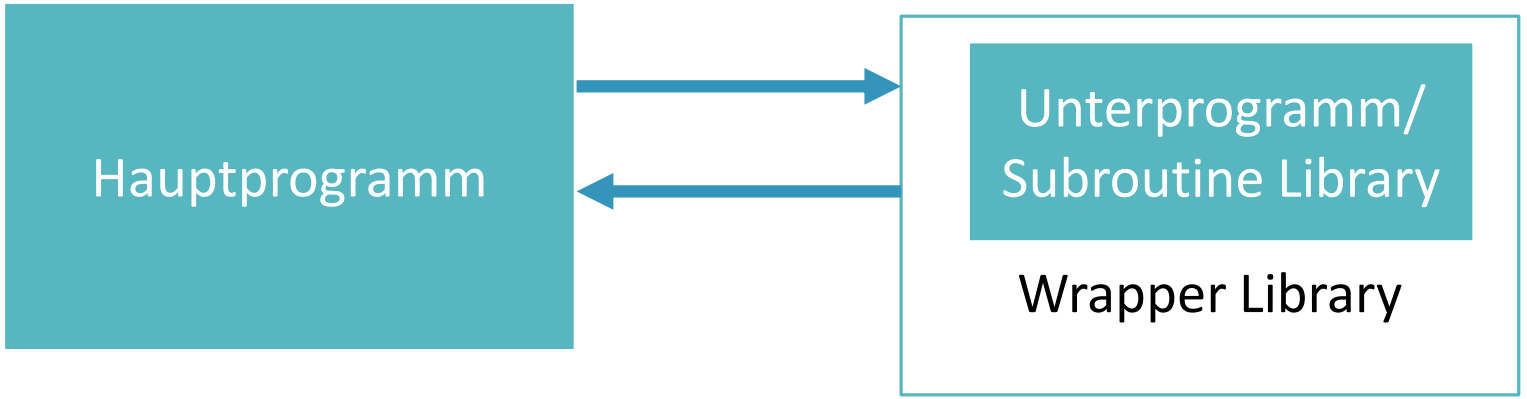


3. Durch die Verwendung einer Bindung von Subroutine Libraries

- Unterprogramm oder Subroutine Library wird an ein vorhandes Programm gebunden
- Verwendung von Wrapper Libraries erleichtern Funktionalität
- Vorteile: - Erfolgreiche Übertragung von Daten zwischen den Programmteilen
  - Bei entsprechender Implementation Aufrufe durch verschiedene Sprachen erlaubt
- Nachteile: - Erzeugung eines zwei-Ebenen Interface
  → keine Rekursion möglich

3. Durch die Verwendung einer Bindung von Subroutine Libraries

- Grundidee:

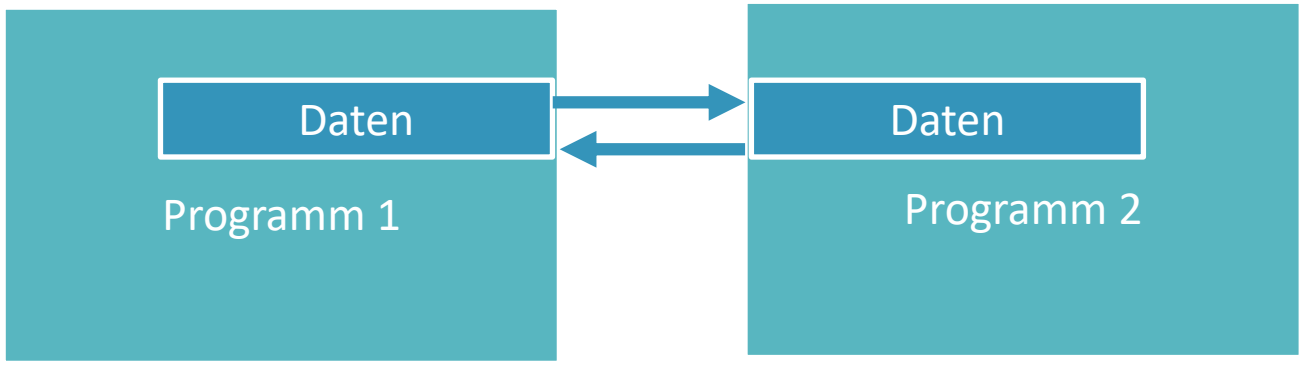


4. Durch die Verwendung von Datentypen einer der Programmiersprachen

- Übertragung von Werten gleichen Datentyps zwischen Unterprogrammen
- Voraussetzung: Gleiche Definition und Behandlung von Datentypen
- Vorteile: Erfolgreiche Übergabe von Werten
- Nachteile: Ungenauigkeiten bei der Übertragung möglich

4. Durch die Verwendung von Datentypen einer der Programmiersprachen

- Grundidee:

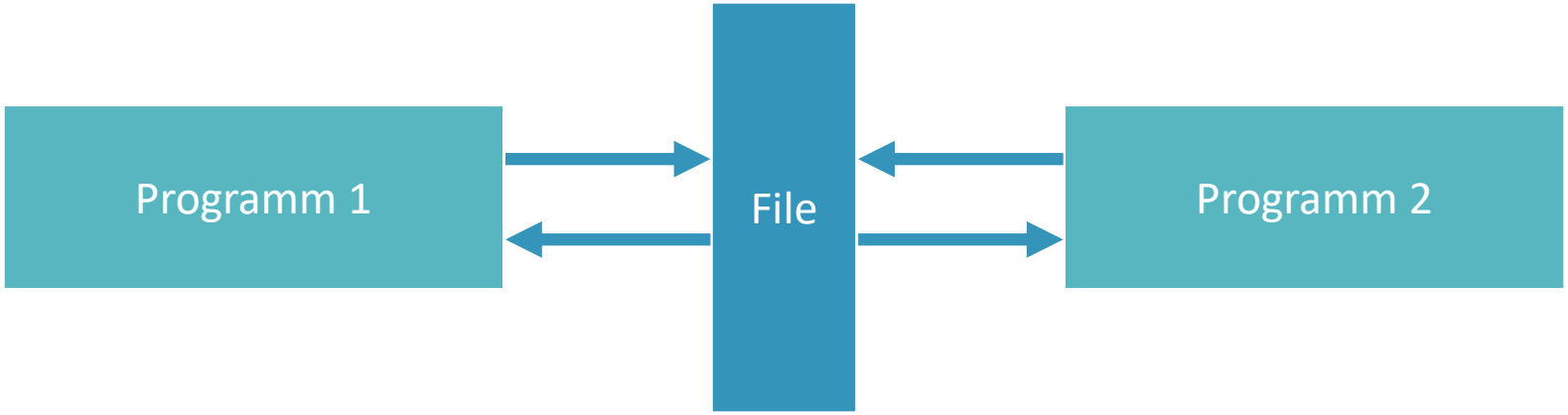


5. Durch die Verwendung einer Kombinierung bereits vollständiger Programme

- Meist verwendete Vorgehensweise
- Zusammenfügung bereits bestehender Programme
- Interaktion durch das Lesen und Schreiben von Daten in Files/Libraries
- Vorraussetzung: Sequentieller Ablauf
- Vorteile: Relativ einfaches Verständnis
- Nachteile: Zu viele Sprünge führen zur Ineffizienz

5. Durch die Verwendung einer Kombinierung bereits vollständiger Programme

- Grundidee:



6. Durch die Verwendung von Tools

- Viele verschiedene Arten: - Tools zur Erzeugung von Libraries
  - Tools zur Verwendung von „Mischsprachen“
  - Tools zur Übersetzung von Datentypen

- Beispiel: SWIG
  - Bindung von C-Code an verschiedene Skriptsprachen
  - Möglichkeit der Erzeugung von Wrapperlibraries
  - Erzeugung von Interfaces

Beispiele für beliebte Sprachenkombinationen

- Python und C/C++ (z.B. Desktopanwendungen)
- PHP und Javascript (z.B. Webanwendungen)
- Fortran und C (hauptsächlich wissenschaftliche Programme)
- R und SQL (Abfrage von Daten aus Datenbanken und die statistische Auswertung dieser)

Beispiel: Aufruf von Fortran-Funktionen in C

- Erstellung eines Fortran-Unterprogramms oder einer -Funktion:

```
function HALLO()
        print *, “Hallo!“
end function HALLO()
```

- Umformatierung des Funktionsaufrufs in C-Format:

```
hallo_()
```

- Deklaration der Funktion in C:

```
void hallo_();
```

Beispiel: Aufruf von Fortran-Funktionen in C

- Aufruf der Funktion in C durch beispielsweise:

```
int main(void)
{
hallo_();
return 0;
}
```

- Kompilieren des Fortran-Unterprogramms/-funktion

```
gfortran -c hallo.f95
```

Beispiel: Aufruf von Fortran-Funktionen in C

- Kompilieren des C-Programms

```
gcc -c  main.c
```

- Zusammenfügen der beiden kompilierten Dateien

```
gfortran -o hw main.o hallo.o
```

Beispiel: Aufruf von C-Funktionen in Fortran

- Funktion in C wird erstellt:

```
#include<stdio.h>

int main(void)
{
printf(“Hallo!“);
return 0;
}
```

Beispiel: Aufruf von C-Funktionen in Fortran

- Die Schnittstelle von Fortran zu C muss in Fortran in einem Modul ausdrücklich als Interface definiert werden

```
MODULE CDEMO
        INTERFACE
            SUBROUTINE main()
            !DEC$ ATTRIBUTES C :: main          // Compiler-Direktive
            END SUBROUTINE
        END INTERFACE
END MODULE
```

Beispiel: C-Erweiterung für Python

- Möglichkeit in Python C-Objekttypen und deren Funktionen zu verwenden (C-Erweiterungsmodul)
- **Allgemeine Vorgehensweise:**
- Von Python ausgehend soll die Funktion „hallo“ aus dem C-Erweiterungsmodul „demo“ aufgerufen werden:

```
>>> import demo
>>> demo.hallo()
```

- Wechseln in C und starten eines Projektes „demomodule.c“

Beispiel: C-Erweiterung für Python

- Um C Zugriff auf die Python-API (und auf die Laufzeitumgebung) zu geben, beginnt der C-Code mit der Zeile:

```
#include<Python.h>
```

(enthält bereits <stdio.h>, <string.h>, <errno.h>, <stdlib.h>)

- Alle für den Nutzer, durch „Python.h“ definierten, Symbole besitzen das Präfix „Py“
- Definition der neuen C-Funktion:

```
static PyObject* demo_hallo( PyObject* self, PyObject *args) {
printf(“Hallo!“)
return 0;
}
```

Zusammenfassung

- Mixed-Language-Programming: Verwendung mehr als einer Programmiersprache
- Sehr viele, sehr verschiedene Möglichkeiten der Implementation

  → Vorraussetzung: Interface – Benutzer – Beziehung
- Verbindung von sehr vielen Sprachen möglich
- Entstehung neuer Programmiersprachen und Tools

Ausblick

- Hypothese:
  - Private, kommerzielle Nutzer:

    - abnehmende Beliebtheit durch Entstehen neuer, schnell erlernbarer Programmiersprachen

  - Wissenschaft:

    - konstante „Beliebtheit“ durch gute Funktionalität bei großen Projekten, Möglichkeit der Einschätzung der Performance, des Speicherplatzes

Quellen

Abbildungen:

- Abb 1. „The structure of Mixed Language Programming Realization“ (1985, B. Einarsson)
- D – Logo https://upload.wikimedia.org/wikipedia/commons/thumb/2/24/D_Programming_Language_logo.svg/1013px-D_Programming_Language_logo.svg.png

Text:

- „The structure of Mixed Language Programming Realization“ (1985, B. Einarsson)

Quellen

- https://www.youtube.com/watch?v=s6cvSkbWG3s&t=207s
- https://stackoverflow.com/questions/17845931/calling-c-function-subroutine-in-fortran-code?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
- https://www.heise.de/ct/ausgabe/2015-18-Die-passende-Programmiersprache-finden-2767703.html
- https://openwsn.atlassian.net/wiki/spaces/OW/pages/8978440/Mixing+Python+and+C
- https://stackoverflow.com/questions/10202306/python-bindings-how-does-it work?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa
- http://www.swig.org/Doc1.3/Introduction.html#Introduction_nn2

Quellen

- https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages
- http://www.swig.org/exec.html
- https://srv.rz.uni-bayreuth.de/lehre/fortran90/vorlesung/V14/V14.html
- https://docs.python.org/2/extending/extending.html
- https://www.youtube.com/watch?v=urcy6-kXZDw
- https://docs.python.org/2/c-api/intro.html
- https://www.computerworld.com/article/2467812/internet/polyglot-programming----development-in-multiple-languages.html
- https://www.wired.com/2014/07/d-programming-language/

Quellen

- https://dlang.org
- https://wiki.dlang.org/Getting_Started