h_da	Embedded SW-Praktikum	
Termin 1	Einarbeitung/Speicher	Prof. Akelbein

1. Funktionen und Speicherverbrauch

Erstellen Sie in der Code::Blocks-IDE des Laborrechners und des Raspberries ein Programm, unterschiedliche Klassen, die das Überladungsprinzip virtual für Funktionen verwenden.

1. Erzeugen Sie eine Instanz einer Klasse mit einer virtuellen Methode. Geben Sie den Speicherbedarf des Objekts aus. Schauen Sie in dem Executable mittels nm, welche Symbole aus dem C++ Programm Sie dort wiederfinden. Fügen Sie weitere virtuelle Methoden hinzu. Ändert sich etwas am Speicherbedarf pro Objekt?
2. Erzeugen Sie eine Klasse, die die Eigenschaften zweier weiterer Klassen erbt, wobei beide Basisklassen jeweils über virtuelle Funktionen verfügen. Welcher Speicherbedarf entsteht pro Objekt? Wie sehen die Symbole im Executable aus?

2. Speicheralignment

Erzeugen Sie einen POD-struct, die aus mehreren Elementen besteht und den Speicher möglichst ungünstig belegt. Geben Sie den Speicherplatzbedarf als Bildschirmausgabe aus. Erzeugen Sie eine POD-Struktur, die die gleichen Komponenten enthält, aber möglichst wenig Speicher verbraucht. Ersetzen Sie den POD-struct durch einen POD-union. Wie ändert sich jeweils der Speicherbedarf auf Laborrechner und Raspberry?

3. Plazierung und Initialisierung

Erzeugen Sie ein Programm bestehend aus zwei Quelltextdateien. Untersuchen Sie mit nm, in welchen Segmenten die Plazierung stattfindet. Untersuchen Sie die Initialisierungs-reihenfolge.

1. Erzeugen Sie in beiden Quelltextdateien globale und statische Variablen, die Sie zum Teil mit Werten initialisieren und zum Teil nicht initialisiert lassen. In welchen Segmenten werden die Variablen platziert?
2. Initialisieren Sie Variablen durch andere Variablen. Welche Werte nehmen die Variablen an?

Erzeugen Sie Klassen, die nur aus einem Konstruktor bestehen, der den Klassennamen ausgibt. Verwenden Sie ein Objekt pro Klasse und zeigen Sie so die Initialisierungs-reihenfolge auf.

1. Erzeugen Sie jeweils zwei globale Klassenobjekte in den beiden Quelltextdateien. In welcher Reihenfolge werden die Klassenobjekte initialisiert? Wie können Sie die Reihenfolge beeinflussen?
2. Erzeugen Sie ein statisches Klassenobjekt in der main(). Schreiben Sie eine zusätzliche Funktion mit einem statischen Klassenobjekt. Wann findet die Initialisierung statt? Erkennen Sie mittels nm, wie eine Mehrfachinitialisierung der statischen Klassenobjekte nach dem ersten Funktionsaufruf verhindert wird?

Tipps:

nm - demangled Names und Größe:

Die Option nm –C gibt Symbole in der Demangled-Form aus, in der sie Klassen- und Funktionsnamen in C++ Notation erkennen können. Die Bedeutung der Buchstaben vor den Symbolen zeigt Ihnen man nm. Mit der Option –S können Sie sich die Größe der Objekte anzeigen lassen.

Optimierung:

Unter Build-Options->Compiler Settings->Other Options können Sie mit der Option –O0 alle Optimierungen ausschalten.

grep:

Um die relevanten Teile aus der nm-Ausgabe zu erhalten, können Sie diese mittels grep durch ein zeilenweises Suchfilter leiten. Wenn Sie ein Suchmuster in den Variablennamen haben, filtert Ihnen grep alle anderen Zeilen heraus.

nm –C –S ./myprogram | grep mypattern

-> enthalten alle Variablen mypattern, sehen Sie nur die Zeilen mit Ihren Variablen

Falls Sie wissen wollen, was noch alles im Binärformat ´elf´ enthalten ist, können Sie sich mit readelf –a ./myprogram alle Inhalte anzeigen lassen, die die ausführbare Datei beschreiben.