Programme, Functions, Subprogramme und Subroutinen

Programme, Functions, Subprogramme und Subroutinen

Dieses Dokument liefert Informationen über die Objekttypen, die als Routinen, d.h. als untergeordnete Programme aufgerufen werden können.

Anmerkung:
Obwohl sie auch von anderen Objekten aufgerufen werden, sind Helproutinen und Maps genau genommen keine Routinen als solche, und werden deshalb in getrennten Dokumenten beschrieben; siehe Abschnitt *Helproutinen*.

Die folgenden Themen werden behandelt:

- Modulare Anwendungsstruktur
- Mehrere Stufen (Levels) aufgerufener Objekte
- Programm
- Function
- Subroutine
- Subprogramm
- Verarbeitungsablauf beim Aufruf eines Unterprogramms

Modulare Anwendungsstruktur

Eine typische Natural-Anwendung besteht nicht aus einem einzigen großen Programm, sondern ist in mehrere Module aufgeteilt. Jedes dieser Module stellt eine funktionale Einheit von überschaubarer Größe dar, und jedes Modul ist mit den anderen Modulen der Anwendung auf eine klar definierte Weise verbunden. Dadurch ergibt sich eine übersichtlich strukturierte Anwendung, was die Entwicklung und anschließende Wartung erheblich erleichtert und beschleunigt.

Ein Hauptprogramm, das ausgeführt wird, kann andere Programme, Subprogramme, Subroutinen, Helproutinen und Maps aufrufen. Diese Objekte können ihrerseits wiederum andere Objekte aufrufen (eine Subroutine kann beispielsweise eine andere Subroutine aufrufen). Dadurch kann die modulare Struktur einer Anwendung äußerst komplex und vielschichtig werden.

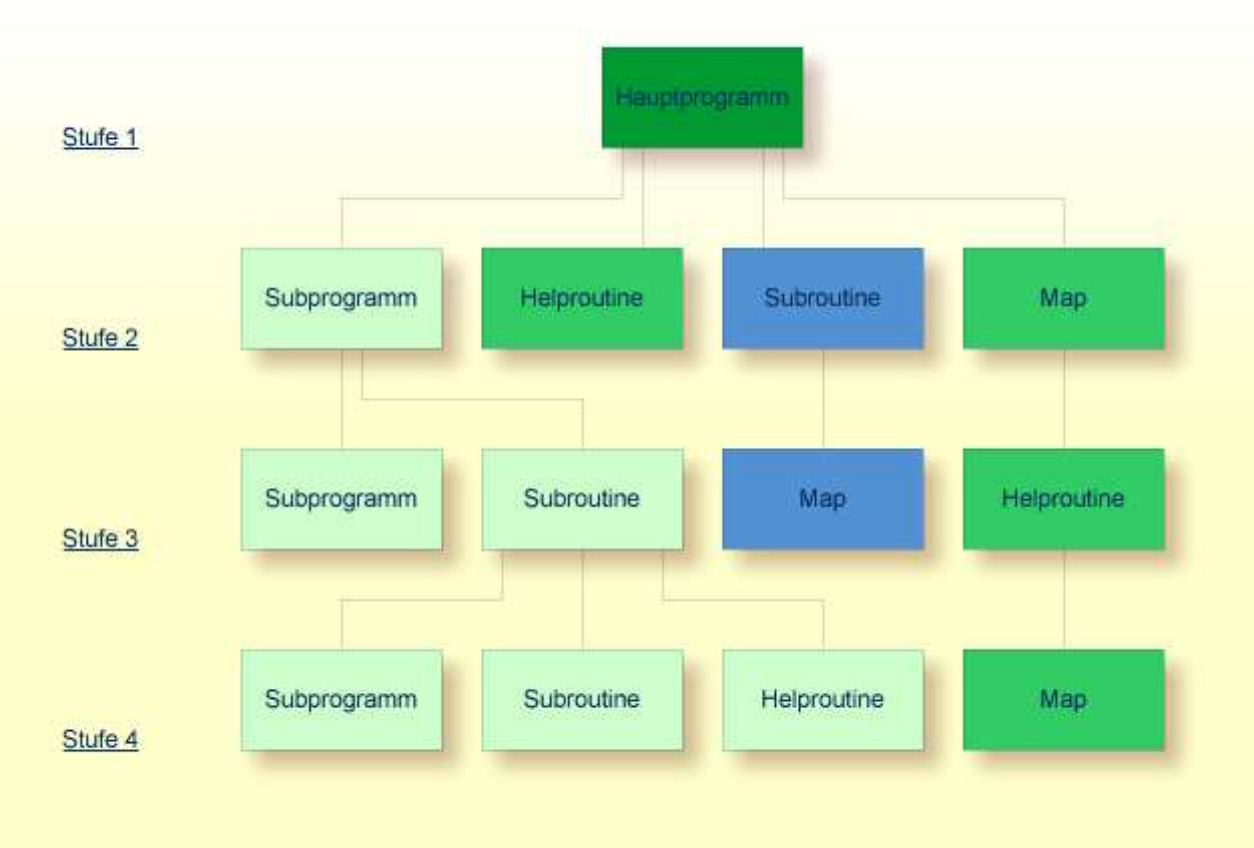
Mehrere Stufen (Levels) aufgerufener Objekte

Ein aufgerufenes Objekt ist jeweils eine Stufe (Level) unter dem Objekt, von dem es aufgerufen wurde; d.h. mit jedem Aufruf eines untergeordneten Objekts erhöht sich die Stufennummer um 1.

Ein Programm, das selbständig ausgeführt wird, wird auf Stufe 1 eingeordnet; Subprogramme, Subroutinen, Maps oder Helproutinen, die direkt von diesem Hauptprogramm aufgerufen werden, sind auf Stufe 2; ruft eine dieser Subroutinen ihrerseits eine andere Subroutine auf, so ist letztere auf Stufe 3.

Wird von einem Objekt über ein FETCH-Statement ein anderes Programm aufgerufen, so wird dies als Hauptprogramm eingestuft und auf Stufe 1 eingeordnet. Ein Programm, das mit FETCH RETURN aufgerufen wird, wird dagegen als Unterprogramm eingestuft und ist eine Stufe unter dem Objekt, von dem es aufgerufen wurde.

Die folgende Abbildung enthält ein Beispiel für mehrere Stufen (Levels) aufgerufener Objekte und zeigt, wie diese Stufen gezählt werden:



Um die Level-Nummer des Objekts, das gerade ausgeführt wird, zu erfahren, können Sie die Systemvariable *LEVEL verwenden (die in der *Systemvariablen*-Dokumentation beschrieben ist).

Der vorliegende Abschnitt behandelt folgende Natural-Objekttypen, die als Unterprogramme (d.h. als untergeordnete Programme) aufgerufen werden können:

- Programm
- Function
- Subroutine
- Subprogramm

Helproutinen und Maps werden zwar auch von anderen Objekten aufgerufen, sind aber keine Unterprogramme im eigentlichen Sinne und werden daher in separaten Abschnitten behandelt, siehe *Helproutinen* und *Maps*.

Grundsätzlich unterscheiden sich Programme, Subprogramme und Subroutinen dadurch voneinander, wie Daten zwischen ihnen übergeben werden können und welche Data Areas sie gemeinsam benutzen können. Die Entscheidung, welchen Objekttyp Sie verwenden, ergibt sich daher im wesentlichen aus der Datenstruktur Ihrer Anwendung.

Programm

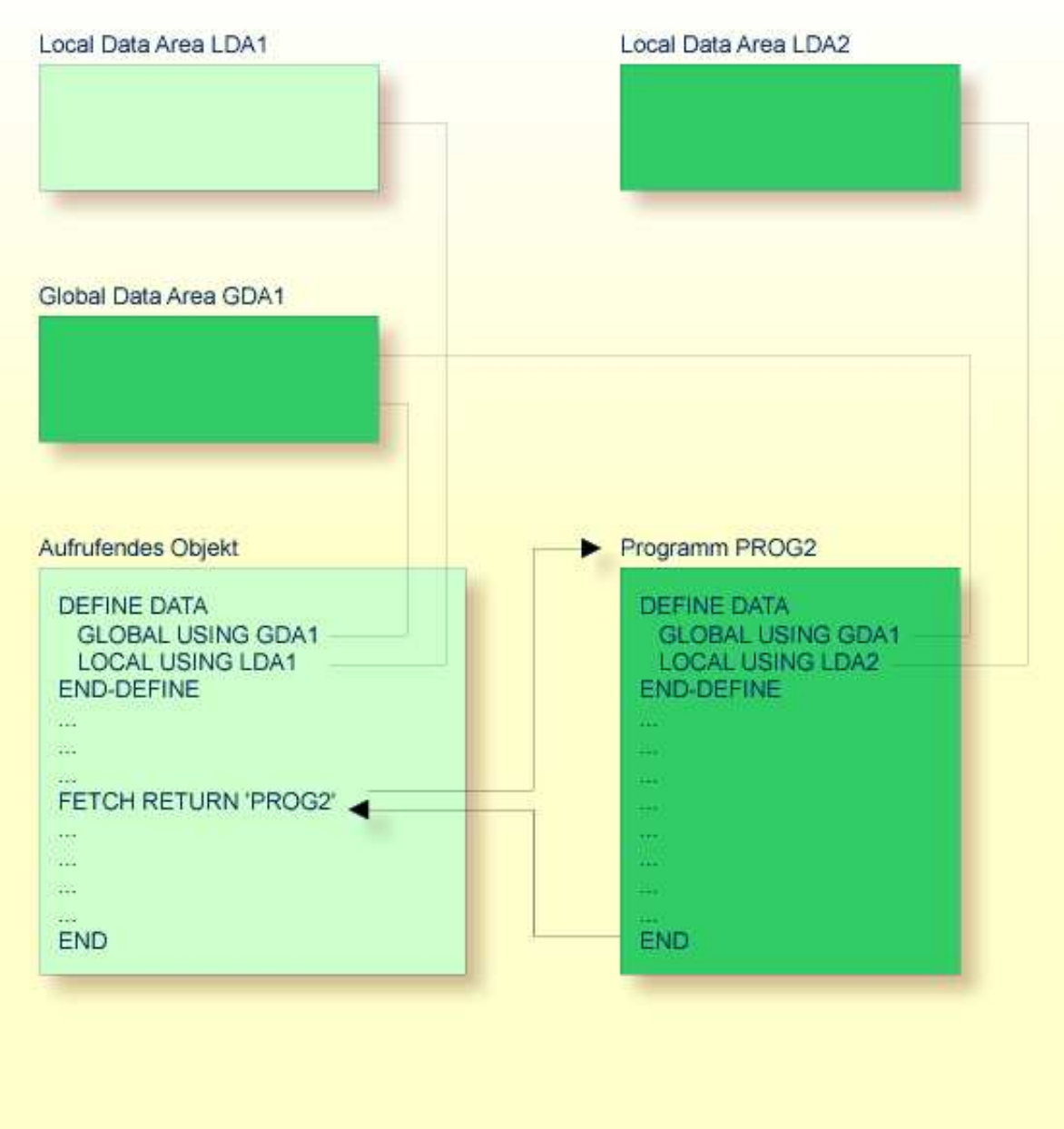
Ein Programm kann selbständig ausgeführt — und getestet — werden.

- Um ein Source-Programm zu kompilieren und anschließend auszuführen, verwenden Sie das Systemkommando `RUN`.
- Um ein Programm auszuführen, das bereits in kompilierter Form existiert, verwenden Sie das Systemkommando `EXECUTE`.

Ein Programm kann von einem anderen Objekt mit einem `FETCH`- oder `FETCH RETURN`-Statement aufgerufen werden. Das aufrufende Objekt kann ein Programm, ein Subprogramm, eine Function, eine Subroutine oder eine Helproutine sein.

- Wenn ein Programm mit `FETCH RETURN` aufgerufen wird, wird die Ausführung des aufrufenden Objekts unterbrochen — nicht beendet —, und das aufgerufene Programm wird als *Unterprogramm* aktiviert. Wenn die Ausführung des aufgerufenen Programms beendet ist, wird das aufrufende Objekt reaktiviert und seine Ausführung mit dem nächsten Statement nach dem `FETCH RETURN`-Statement fortgesetzt.
- Wenn ein Programm mit `FETCH` aufgerufen wird, wird die Ausführung des aufrufenden Objekts beendet, und das aufgerufene Programm wird als *Hauptprogramm* aktiviert. Das aufrufende Objekt wird nach beendeter Ausführung des aufgerufenen Programms nicht reaktiviert.

Mit FETCH RETURN aufgerufenes Programm

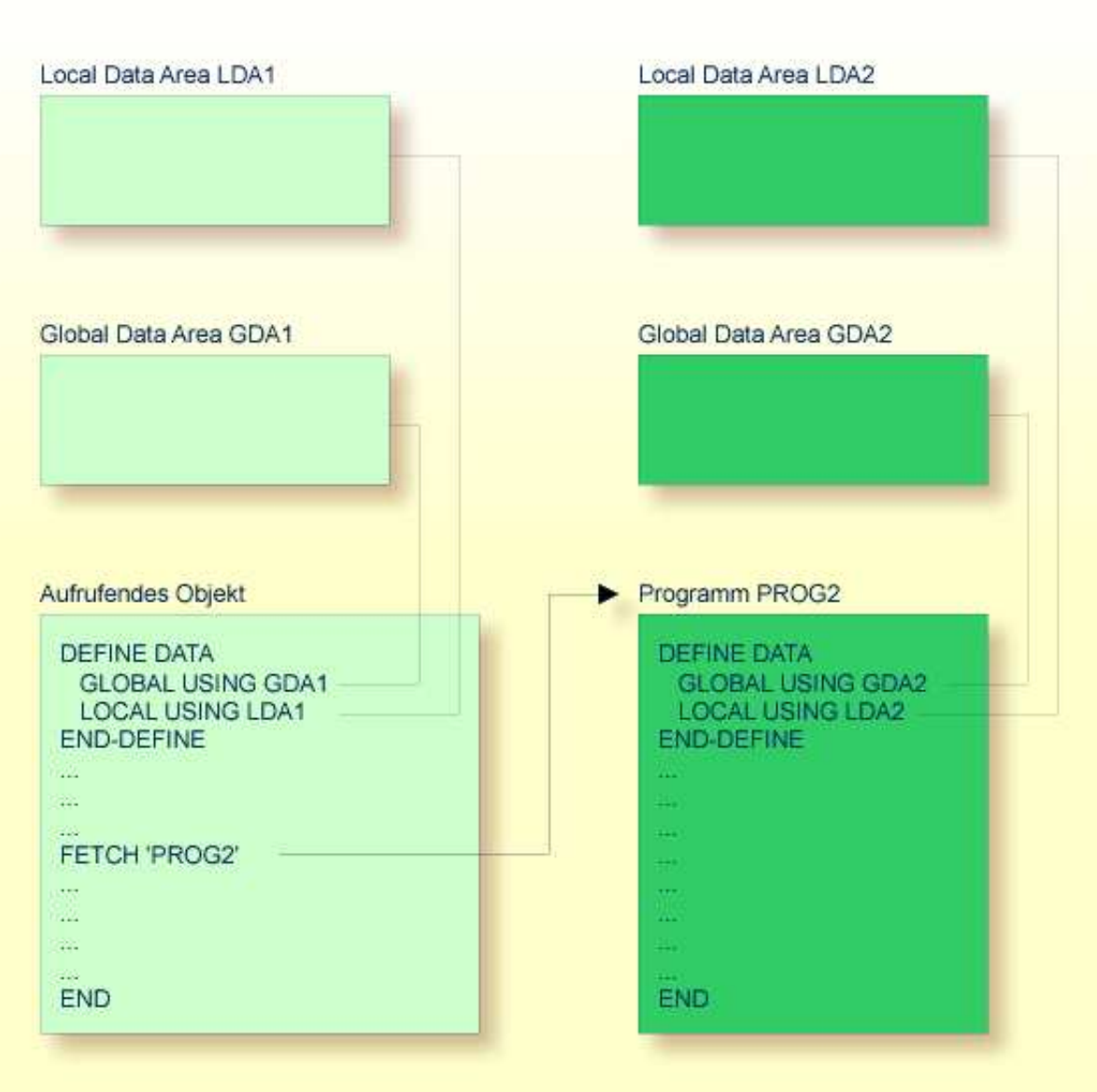


Ein mit FETCH RETURN aufgerufenes Programm kann auf die vom aufrufenden Objekt benutzte Global Data Area zugreifen.

Darüber hinaus kann jedes Programm seine eigene Local Data Area haben, in der die nur in diesem Programm verwendeten Felder definiert sind.

Ein mit FETCH RETURN aufgerufenes Programm kann jedoch keine eigene Global Data Area haben.

Mit FETCH aufgerufenes Programm



Ein mit FETCH als Hauptprogramm aufgerufenes Programm verwendet in der Regel seine eigene Global Data Area (wie in der obigen Abbildung gezeigt). Es könnte allerdings auch dieselbe Global Data Area verwenden wie das aufrufende Objekt.

Anmerkung:
Ein Source-Programm kann auch mit einem RUN-Statement aufgerufen werden; siehe RUN-Statement im der *Statements*-Dokumentation.

Function

Dieses Dokument beschreibt die Vorteile der Verwendung des Programmierobjekts "Function", erläutert die Unterschiede bei der Verwendung von "Function Calls" und "Subprogram Calls" und stellt die verfügbaren Methoden zum Definieren und Aufrufen einer "Function" vor.

Ein Objekt des Typs "Function" enthält die Definitionen einer einzelnen Funktion und kann wie im folgenden Code-Beispiel gezeigt aufgebaut sein:

```
DEFINE FUNCTION
   ...
   DEFINE SUBROUTINE
   ...
   END-SUBROUTINE
   ...
END-FUNCTION
```

Der Statements-Block zwischen `DEFINE FUNCTION` und `END-FUNCTION` muss alle Statements enthalten, die ausgeführt werden sollen, wenn die Function aufgrufen wird.

Innerhalb der Function-Definition dürfen auch interne Subroutinen defineirt werden.

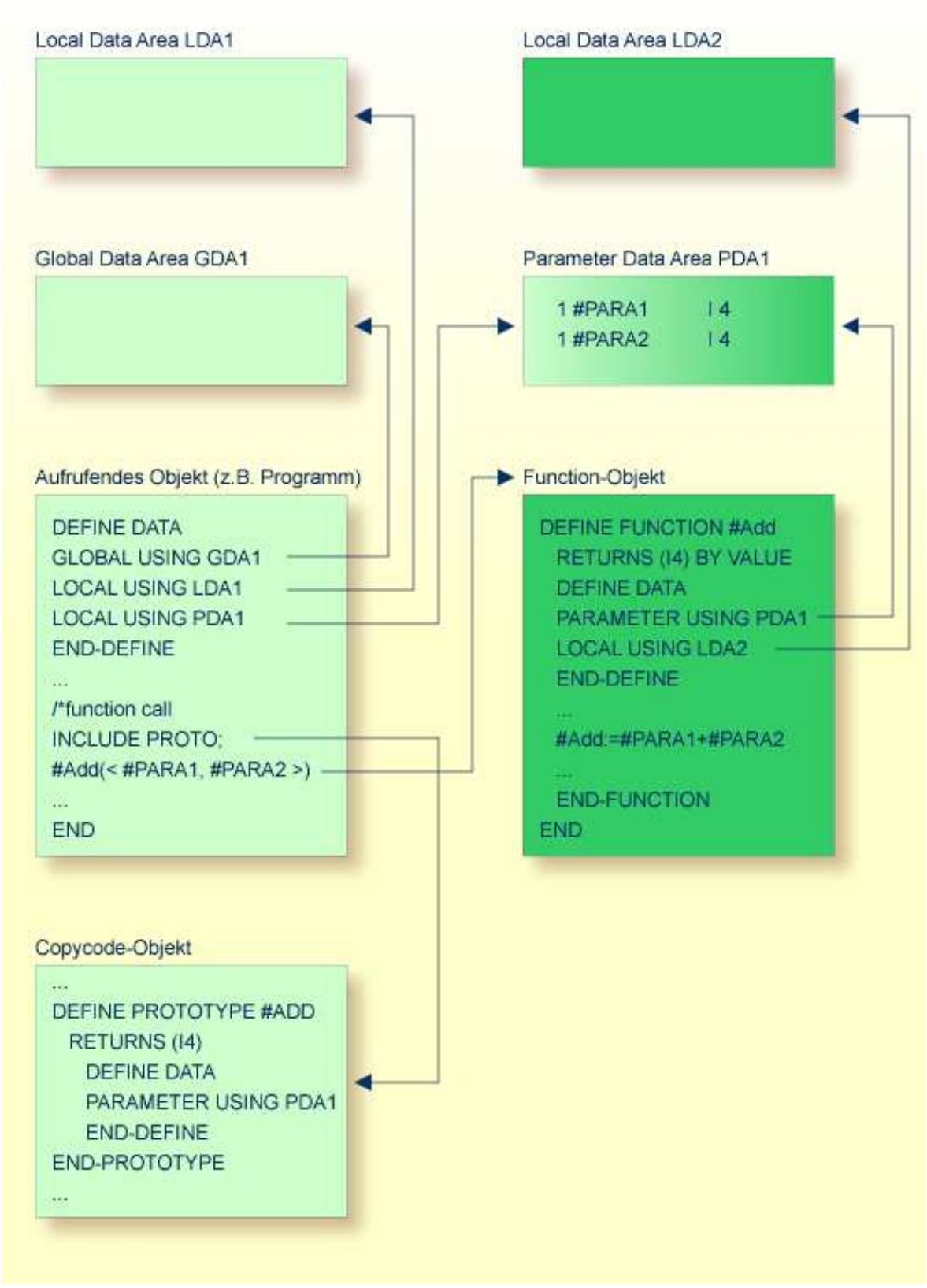
Zum Aufrufen einer Function wird die Function Call-Syntax verwendet.

Wenn Sie eine Code-Block haben, der innerhlab des Objekts mehrmals ausgeführt werden soll, ist es von Nutzen, eine interne Subroutine zu verwenden. Dann brauchen Sie diesen Block nur einmal in einem `DEFINE SUBROUTINE`-Statement-Bock zu codieren und können ihn mit mehreren `PERFORM`-Statements aufrufen.

Die Global Data Area des aufrufenden Objekts (z.B. `GDA1`) kann nicht in der Function-Definition referenziert werden. Darüber hinaus können Objekte, die von einer Function aufgerufen werden, nicht die Global Data Area (GDA) des die Function aufrufenden Objekts (`GAD1`) aufrufen, weil die Laufzeitumgebung zu Beginn einer Function eine neue Global Data Area erstellt.

Parameter Data Areas (z.B. `PDA1`) können Sie benutzen, um auf Parameter für Function Calls und auf Function-Definitionen zuzugreifen, um so den Pflegeaufwand zu minimieren, falls die Parameter geändert werden müssen.

Das Copycode-Objekt, das die Prototype-Definition enthält, wird nur beim Kompilieren benutzt, um den Typ der Return-Variablen für die Function-Call-Referenz zu bestimmen und um, falls gewünscht, die Parameter zu prüfen.


Local Data Area LDA1
Local Data Area LDA2
Global Data Area GDA1
Parameter Data Area PDA1
1 #PARA1 I 4
1 #PARA2 I 4
Aufrufendes Objekt (z.B. Programm)
DEFINE DATA
GLOBAL USING GDA1
LOCAL USING LDA1
LOCAL USING PDA1
END-DEFINE
...
/*function call
INCLUDE PROTO;
#Add(< #PARA1, #PARA2 >)
...
END
Function-Objekt
DEFINE FUNCTION #Add
RETURNS (I4) BY VALUE
DEFINE DATA
PARAMETER USING PDA1
LOCAL USING LDA2
END-DEFINE
...
#Add:=#PARA1+#PARA2
...
END-FUNCTION
END
Copycode-Objekt
...
DEFINE PROTOTYPE #ADD
RETURNS (I4)
DEFINE DATA
PARAMETER USING PDA1
END-DEFINE
END-PROTOTYPE
...

Subroutine

Die Statements, aus denen eine Subroutine besteht, müssen innerhalb eines `DEFINE SUBROUTINE` ... `END–SUBROUTINE`-Statement-Blocks definiert werden.

Eine Subroutine wird mit einem `PERFORM`-Statement aufgerufen.

Eine Subroutine kann eine *interne Subroutine* oder eine *externe Subroutine* sein:

- **Interne Subroutine**
  Eine interne Subroutine wird innerhalb des Objekts, das das sie aufrufende `PERFORM`-Statement enthält, definiert.

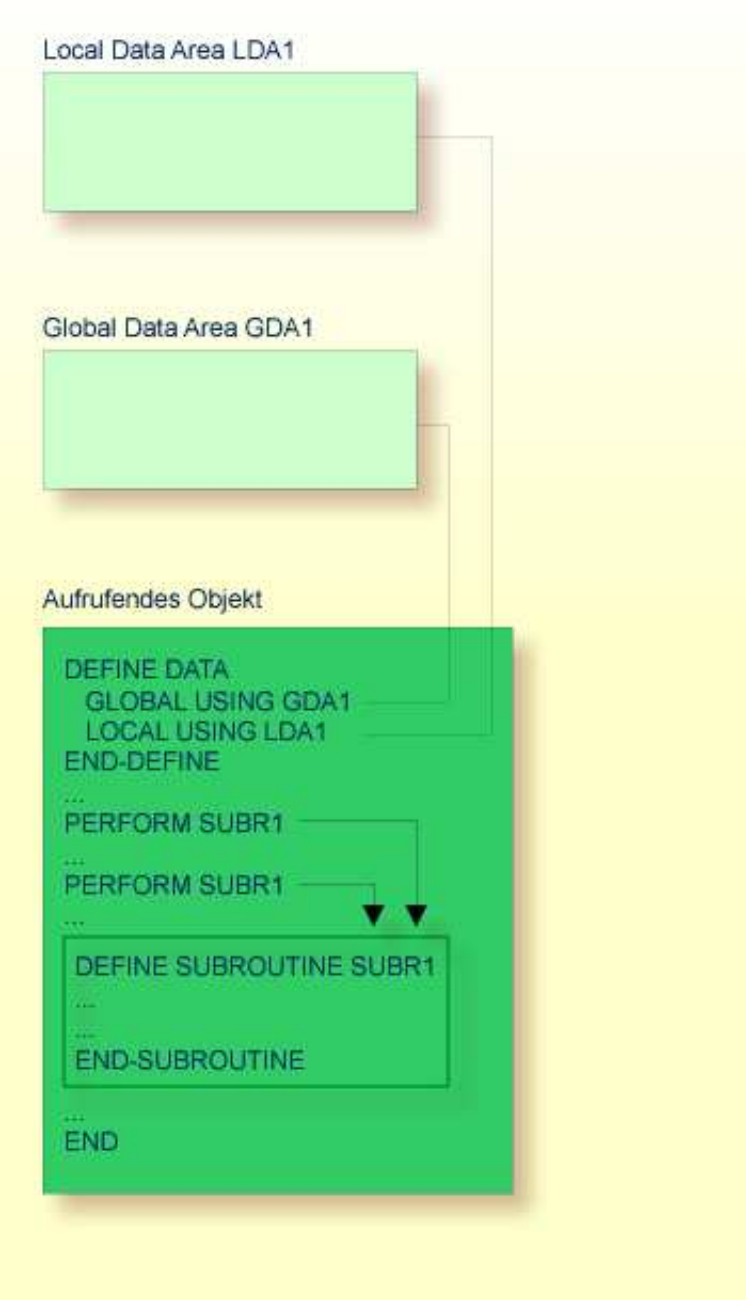
- **Externe Subroutine**
  Eine externe Subroutine wird als separates Objekt - vom Typ Subroutine - außerhalb des Objektes, das sie aufruft, definiert.

Falls Sie einen Code-Block haben, der innerhalb eines Objekts mehrmals ausgeführt werden soll, ist es sinnvoll, eine interne Subroutine zu verwenden. Sie müssen diesen Block dann nur einmal innerhalb eines `DEFINE SUBROUTINE`-Statement-Blocks kodieren, und rufen ihn dann mit mehreren `PERFORM`-Statements auf.

Die folgenden Themen werden nachfolgend erörtert:

- Interne Subroutine
- Welche Daten einer internen Subroutine zur Verfügung stehen
- Externe Subroutine
- Welche Daten einer externen Subroutine zur Verfügung stehen

Interne Subroutine



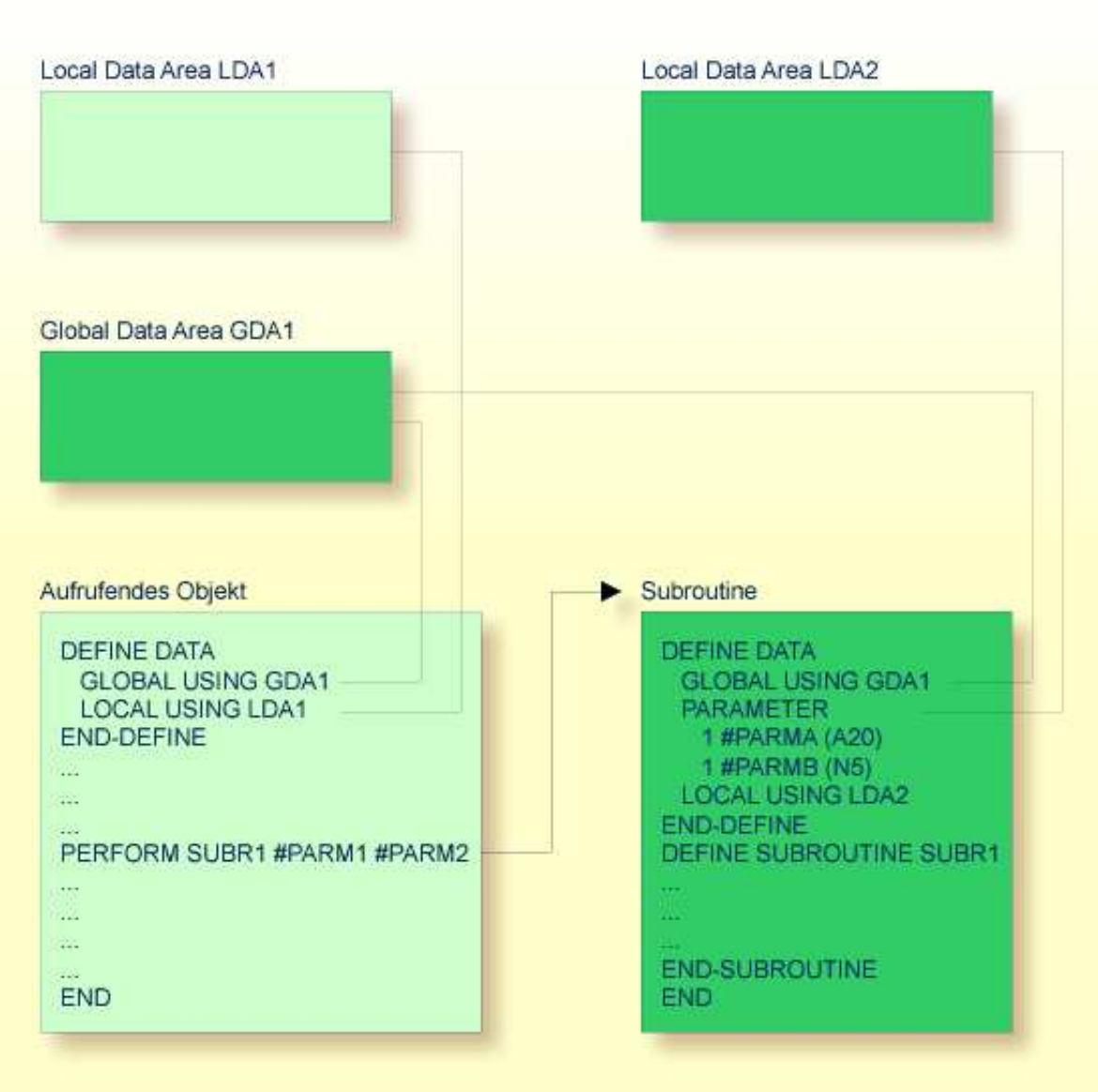
Eine interne Subroutine kann in einem Programmierobjekt vom Typ Programm, eine Function, Subprogramm, Subroutine oder Helproutine enthalten sein.

Wenn eine interne Subroutine so groß ist, dass sie die Lesbarkeit des Objekts, in dem sie enthalten ist, beeinträchtigt, kann es ratsam sein, sie in einer externen Subroutine unterzubringen, um die Lesbarkeit der Anwendung zu verbessern.

Welche Daten einer internen Subroutine zur Verfügung stehen

Eine interne Subroutine kann auf die Local Data Area und die Global Data Area des Objektes, in dem sie enthalten ist, zugreifen.

Externe Subroutine



Eine externe Subroutine — also ein *Objekt* vom Typ Subroutine — kann nicht selbständig ausgeführt werden. Sie muss von einem anderen Objekt aufgerufen werden. Das aufrufende Objekt kann ein Programm, eine Function, ein Subprogramm, eine Subroutine oder eine Helproutine sein.

Welche Daten einer externen Subroutine zur Verfügung stehen

Eine externe Subroutine kann auf die Global Data Area des aufrufenden Objekts zugreifen.

Darüber hinaus können mit dem PERFORM-Statement Parameter von dem aufrufenden Objekt an die externe Subroutine übergeben werden. Diese Parameter müssen entweder im DEFINE DATA PARAMETER-Statement der Subroutine oder in einer von der Subroutine genutzen Parameter Data Area definiert werden.

Außerdem kann eine externe Subroutine eine eigene Local Data Area haben, in der die Felder definiert sind, die nur innerhalb der Subroutine verwendet werden.

Eine externe Subroutine kann jedoch keine eigene Global Data Area haben.

Subprogramm

Ein Subprogramm enthält in der Regel eine allgemein verfügbare Standardfunktion, die von verschiedenen Objekten in einer Anwendung benutzt wird.

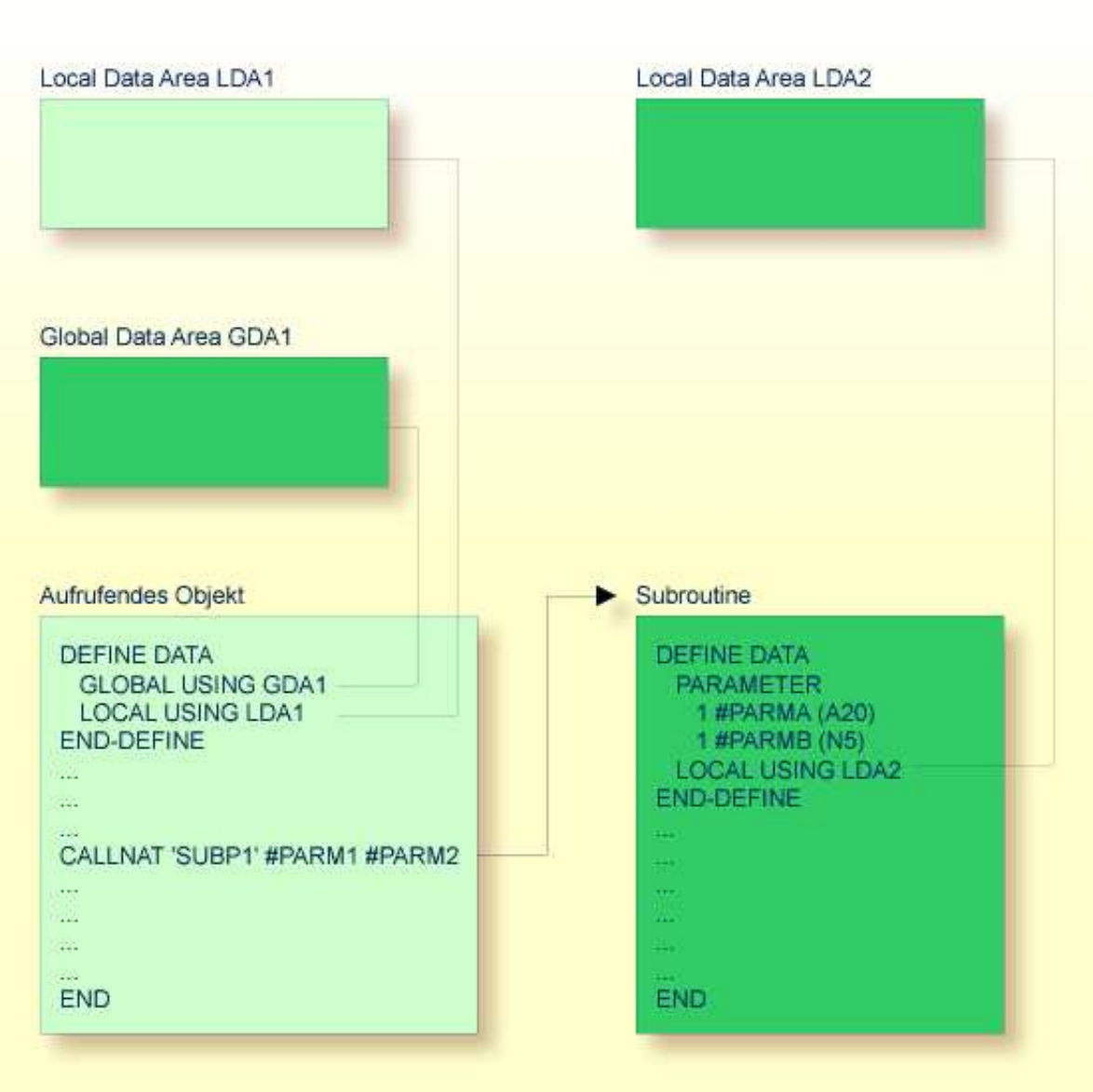
Ein Subprogramm kann nicht selbständig ausgeführt werden. Es muss von einem anderen Objekt aufgerufen werden. Das aufrufende Objekt kann ein Programm, eine Function, ein Subprogramm, eine Subroutine oder eine Helproutine sein.

Ein Subprogramm wird mit einem CALLNAT-Statement aufgerufen.

Wenn das CALLNAT-Statement ausgeführt wird, wird die Ausführung des aufrufenden Objekts unterbrochen und das Subprogramm ausgeführt. Nach der Ausführung des Subprogramms wird die Ausführung des aufrufenden Objekts mit dem nächsten Statement nach dem CALLNAT-Statement fortgesetzt.

Welche Daten einem Subprogramm zur Verfügung stehen

Mit dem CALLNAT-Statement können Parameter von dem aufrufenden Objekt an das Subprogramm übergeben werden. Diese Parameter sind die einzigen Daten, die dem Subprogramm vom aufrufenden Objekt zur Verfügung stehen. Sie müssen entweder im DEFINE DATA PARAMETER-Statement des Subprogramms oder in einer vom Subprogramm genutzen Parameter Data Area definiert werden.



Außerdem kann ein Subprogramm eine eigene Local Data Area haben, in der die Felder definiert sind, die innerhalb des Subprogramms verwendet werden.

Wenn ein Subprogramm seinerseits eine Subroutine oder Helproutine aufruft, kann es eine eigene Global Data Area haben und diese gemeinsam mit der Subroutine/Helproutine nutzen.

Verarbeitungsablauf beim Aufruf eines Unterprogramms

Wenn ein `CALLNAT`-, `PERFORM`- oder `FETCH RETURN`-Statement, das ein Unterprogramm — ein Subprogram, eine externe Subroutine bzw. ein Programm — aufruft, ausgeführt wird, wird die Ausführung des aufrufenden Objekts unterbrochen, und die Ausführung des Unterprogramms beginnt.

Die Ausführung des Unterprogramms wird fortgesetzt, bis entweder sein `END`-Statement erreicht ist oder die Verarbeitung des Unterprogramms durch die Ausführung eines `ESCAPE ROUTINE`-Statements gestoppt wird.

In beiden Fällen wird die Verarbeitung des aufrufenden Objekts mit dem nächsten Statement nach dem CALLNAT-, PERFORM- bzw. FETCH RETURN-Statement, mit dem das Unterprogramm aufgerufen wurde, fortgesetzt.

Beispiel: