Anhang VI:

Glossar

Glossar

Ablaufverfolgung

oder Trace wird ein Durchlauf durch ein Programm genannt, wobei jeweils der veränderte Datenspeicher inspiziert wird. Ein Trace kann am Schreibtisch erfolgen (Walkthrough genannt). Im allgemeinen stellt ein Programmiersystem eine Werkzeugunterstützung zur Verfügung.

Abstraktes Datenobjektmodul

ein Modul, der in der Architektur eines Softwaresystems ein abstraktes Datenobjekt in Form eines Datenabstraktions-Bausteins verkapselt.

Abstraktes Datentypmodul

Modul der Architektur, mit dessen Hilfe abstrakte Datenobjekte zur Laufzeit eines Programms erzeugt werden können. In C++ oder Modula-2 erfolgt die Erzeugung über eine Erzeugungsanweisung. Dabei entstehen Verweise auf erzeugte Objekte (abstrakte Datentypmodule mit Zeigersemantik). Diese Handhabung sollte an der Schnittstelle des adt-Moduls nicht sichtbar sein.

Äquivalenz von Programmen

Zwei Programme in einer Programmiersprache heißen stark äquivalent, wenn die Programmiersprachen-Maschine bei beliebiger Programmausführung die gleichen internen Zustände durchläuft. Daraus ergibt sich, daß beide Programme nur triviale Unterschiede besitzen dürfen. Bei schwacher Äquivalenz muß das Ein-/Ausgabeverhalten jeder Berechnung übereinstimmen, intern können beliebige andere Zustände auftreten. Eine effizientere Fassung eines Programms ist zu der ursprünglichen somit schwach äquivalent.

Aggregate

dienen dem direkten Hinschreiben eines konstanten Werts in einem Programm (zusammengesetztes Literal). In C++ gibt es Aggregate nur für Felder. In Modula–2 gibt es keine Aggregate.

Aktualparameter

→ Parameterzuordnung

(Allg.) Benutzbarkeitsbeziehung

Festlegung, daß für die Realisierung eines Bausteins ein anderer Baustein genutzt werden soll (Importe). In Modula-2 werden dabei sogar einzelne Ressourcen dieses Bausteins importiert, in C++ immer der ganze Baustein.

Anweisungen

Anweisungen dienen der Strukturierung der Berechnung (des Anweisungsteils) eines Programmes oder eines Programmbausteins. Durch Benutzung verschiedener Anweisungsformen (→ Kontrollstrukuren) können beliebige Abläufe zur Laufzeit festgelegt werden. Es ist darauf zu achten, daß die statische Festlegung des Anweisungsteils möglichst die Ausführungsreihenfolge wiedergibt (sparsame Verwendung von Sprüngen, übersichtliche Kontrollstrukturen). Anweisungen werden unterschieden in einfache Anweisungen (Wertzuweisung, Prozeduraufruf) und zusammengesetzte Anweisungen (→ Kontrollstrukturen).

Arrays → Felder

Aufzählungsdatentypen

selbstdefinierte oder vordefinierte Datentypen (in C++ `char`, CHAR in Modula-2) in denen sämtliche Werte des Datentyps explizit aufgezählt werden; mit enum in C++.

Ausdrücke

Die Auswertung von Ausdrücken liefert zur Programmlaufzeit Werte. Diese dienen dazu, Variable zu verändern (Zuweisung), Rückgabewerte zu berechnen (Rücksprunganweisung), Eingabewerte für Unterprogramme festzulegen usw. Eine Spezialform von Ausdrücken sind Boolesche Ausdrücke, mit denen ein Teil des Kontrollflusses gesteuert wird. Relationale Ausdrücke sind wiederum ein Spezialfall Boolescher Ausdrücke. Bezüglich der Syntax besitzen Ausdrücke zum einen eine Aufbausyntax (z.B. muß ein binärer Ausdruck zwei Teilausdrücke besitzen). Zum anderen geben kontextsensitive Regeln weitere Restriktionen vor (z.B., daß beide Teilausdrücke vom gleichen Typ sein müssen).

Ausgangs(gabe)parameter

→ Parameterzuordnung

Auswahlanweisung

Sie ist eine Form der Fallunterscheidung (→ Kontrollstrukturen, → Anweisungen) und dient der Unterscheidung verschiedener Fälle, wobei die Auswahl der alternativen Anweisungsfolgen durch Aufzählen aller entsprechenden Werte für eine Anweisungsfolge vorgenommen wird. Diese Werte werden in Auswahllisten angegeben. Werte können einzeln aufgezählt werden sowie durch Unterbereiche angegeben werden. Alle möglichen Werte des Auswahlausdrucks sollten in den Auswahllisten erscheinen.

Bäume

Bäume oder Wurzelbäume sind spezielle Graphen (Datenstrukturen), bei denen jeder Knoten bis auf die Wurzel genau einen Vorgänger besitzt. Wir unterscheiden beliebige n-äre Bäume von binären Bäumen (jeder Knoten hat maximal 2 Nachfolger). Binäre Bäume werden zum Suchen und inkrementellen Sortieren eingesetzt (→ binäre Suchbäume). Zu jedem n-ären Baum gibt es einen binären (sogen. Leftmost–child–right–sibbling–Darstellung). Bäume können knotenorientiert, mit Knotenelementen realisiert werden, die Verweise auf die Nachfolger enthalten. Alternativ dazu können für die Kantendarstellung eigenständige Kantenelemente eingeführt werden (für alle auslaufenden Kanten je ein Kantenelement oder für alle einlaufenden).

bedingte Anweisung

Sie ist eine Form der Fallunterscheidungen (→ Kontrollstrukturen, → Anweisungen) und dient der Programmierung verschiedener Anweisungsfolgen, die bei Zutreffen einer Booleschen Bedingungen ausgeführt werden. Spezialfälle sind einseitig und zweiseitig bedingte Anweisungen. In der allgemein bedingten Anweisung mit `ELSIF` in Modula-2 können beliebig viele Alternativen auftreten. Diese allgemeine bedingte Anweisung muß in C++ durch geschachtelte zweiseitig bedingte Anweisungen simuliert werden.
(`else if`)

bedingte Schleifen

Formen von Schleifen (→ Kontrollstrukturen, → Anweisung), die durch einen Booleschen Ausdruck gesteuert werden. While–Schleifen machen die Prüfung auf wiederholte Ausführung des Schleifenrumpfs am Anfang, until–Schleifen nach Ausführung des Schleifenrumpfes. Für bedingte Schleifen müssen Terminationsüberlegungen angestellt werden, damit ein Programm nicht in eine Endlosschleife gerät. Der Boolesche Ausdruck muß im Schleifenrumpf verändert werden, da sonst eine Endlosschleife oder eine triviale Schleife entsteht (nicht ausgeführt, nur einmal ausgeführt).

binäre Suchbäume

Binäre Suchbäume sind binäre Bäume, bei denen jeder Knoten einen Suchschlüssel enthält. Sie sind so angeordnet, daß der linke Teilbaum eines Knotens nur kleinere und der rechte nur größere Schlüssel als der Knoten enthält. Im Falle eines ausgeglichenen Suchbaums ergibt sich so eine sehr schnelle Suche (O(log n)). Zur Vermeidung von Unbalancierung werden verschiedene Formen ausgeglichener Bäume definiert. Nach jeder Baumänderungsoperation oder nach Auftreten größerer Unbalanciertheit wird der Baum ausgeglichen.

Bit-Operatoren

In C++ für ganzzahlige Datenobjekte definiert. In Modula-2 existieren diese für die Teilmengenhandhabung (→ charakteristische Darstellung) durch Mengenoperatoren, die auf Bit-Vektor-Operationen zurückgespielt werden.

Black – Box – Test

Modultest nur unter Betrachtung der Export-Schnittstelle eines Moduls. Dieser Test berücksichtigt somit nicht die interne Realisierung des Bausteins. Für den Black-Box-Test ist eine saubere Gestaltung der Schnittstelle zwingend erforderlich. Es gibt spezielle Methoden, die die Testfälle für Ein-/Ausgabedaten aus den Wertebereichen ihrer Bestandteile zusammensetzen.

Blockstruktur

Unterprogramme, in manchen Sprachen auch Blöcke, können ineinander geschachtelt werden. Hierbei können lokale Berechnungen eingeführt werden. Die eingeführten Deklarationen haben einen Gültigkeitsbereich (→ Gültigkeitsbereich) und einen Sichtbarkeitsbereich (→ Sichtbarkeitsbereich). Schachtelung gibt es in Modula–2 nur auf der Ebene von Unterprogrammen, in C++ nur mit Blöcken. Die Schachtelung von Modulen in Modula–2 ist wertlos, da sie nicht mit Hilfsmitteln zur getrennten Bearbeitung verbunden ist.

Boolescher Datentyp

Datentyp `bool` (in Erinnerung an den Mathematiker Boole) in C++ für zwei Werte, nämlich wahr (true) oder falsch (false).

Bubblesort

Direktes Sortierverfahren nach der Strategie ”Sortieren durch Vertauschen”.

C++

Erweiterung der Sprache C, die bereits in den 60er Jahren festgelegt wurde (um Objektorientierung). C++ hat eine weite Verbreitung in der industriellen Praxis. C bzw. C++ sind implementierungsnah gestaltet und erlauben damit das Schreiben effizienter Programme.

Call by Reference

(Aufruf über die Adresse) → Parameterzuordnungs-Mechanismus, → Referenzen

Call by Value

(Aufruf über den Wert) → Parameterzuordnungs-Mechanismus

Datenabstraktionsprinzip

Das Datenabstraktionsprinzip beinhaltet (a) Zugriffsoperationen auf logische Sicht nach außen, (b) Verkapselung der Realisierungsdetails von Modulen, die Daten (Zustand, Gedächtnis) handhaben, innerhalb eines Moduls. Das Zerlegen von Softwaresystemen unter Beachtung dieses Prinzips nennt man objektbasierte Softwarekonstruktion oder objektbasiertes Entwerfen. Dieses ist eine Vorstufe der objektorientierten Konstruktion. Die Beachtung des Datenabstraktionsprinzips ist Voraussetzung für änderbare Softwaresysteme (s. Jahr-2000-Problem). Alle komplexen Daten sind in einem Softwaresystem als Bausteine vertreten.

Datentypkonstruktoren

Konstrukte (→ Konstrukte von Programmiersprachen) zur Festlegung der Datenstrukturen eines Programms. Hierzu zählen das Bilden von Feldern, Verbunden, Zeigern, Mengen etc. Datentypkonstruktoren sind ineinander einsetzbar, z.B. Feld von Verbunden, die Komponenten des Verbundes wiederum beliebig strukturiert. Diese Einsetzbarkeit gilt allerdings nicht für alle Datentypkonstruktoren. So darf z.B. in Modula-2 der Datentypkonstruktur `SET OF` nur auf skalare Datentypen (Aufzählungsdatentypen, ganzzahlige Datentypen, Unterbereiche hiervon) angewendet werden.

Deklarationen

Deklarationen dienen der Festlegung zu verwendender Konstanten, Typen, Datenobjekte, Unterprogramme und Module. Deklarationen für Konstanten, Typen und Datenobjekte stehen im Deklarationsteil einer Programmeinheit (Unterprogramm, Modul, Klasse). Deklarationen müssen in Modula-2 vorab stehen, in C++ sollten sie vorab stehen. Die zusätzliche Festlegung durch Deklarationen erlaubt eine Reihe von Überprüfungen zur Compilezeit. Durch Konstantendeklarationen wird ein Bezeichner für einen compilezeitbestimmten Wert eingeführt, eine Typdeklaration führt einen Namen für eine Art von Objekten ein, deren Struktur festgelegt wird, eine Objektdeklaration führt eine Variable eines bestimmten Typs ein, eine Unterprogrammdeklaration führt die Schnittstelle eines Unterprogramms ein.

Direktes Auswählen

Direktes Sortierverfahren nach der Strategie des ”Sortierens durch Auswählen”. (selectionsort)

Direktes Einfügen

Direktes Sortierverfahren nach der Strategie des ”Sortieren durch Einfügen”. (insertionsort)

Dynamischer Speicherbereich

setzt sich aus → Laufzeitkeller und → Halde zusammen.

EBNF

erweiterte Backus–Naur–Form (→ Syntaxnotationen) → Anhang I

Eingangs(gabe)parameter

→ Parameterzuordnung

Endlosschleife

Eine Form von Schleifen (→ Kontrollstrukturen, → Anweisungen). Hier muß insbesondere auf Termination geachtet werden. Die Schleife muß durch einen Sprung (break in C++) verlassen werden. Dabei steuert ein Boolescher Ausdruck das Verlassen der Schleife. Dieser Boolesche Ausdruck muß durch den Schleifenrumpf verändert werden.

Feldaggregat

Ein Feldaggregat ist ein zusammengesetztes Literal, mit dem ein Wert eines Feldes im Programm direkt hingeschrieben werden kann (wie ein Literal für einen skalaren Datentyp). Dies vermeidet die Initialisierung von Feldern über Schleifen, wenn die Komponentenwerte statisch sind.

Felder (Reihungen, engl. Arrays)

Zusammensetzung gleichartiger Datenobjekte zu einem Ganzen. Der Zugriff erfolgt zur Laufzeit mit einem Indexausdruck. Man unterscheidet eindimensionale Felder und mehrdimensionale Felder. Ebenso wird zwischen statischen Feldern (Größe zur Programmerstellungszeit bekannt) sowie dynamischen Feldern (Größe laufzeitabhängig) unterschieden.

Feldtypen mit unspezifizierten Grenzen

→ offene Felder

Flußdiagramm

Graphische Darstellung für ein Programm. Dabei wird nur die Ablaufkontrolle dargestellt. Kontrollstrukturen gibt es nur auf primitiver Ebene (bedingte, unbedingte Sprünge sowie Unterprogrammsprung). Die Form der

graphischen Elemente ist genormt (DIN 66001). → Anhang II

Formalparameter

→ Parameterzuordnung

Funktionsmodul

Zusammenfassung verschiedener Funktionen in einem Modul. Diese sollten dabei das gleiche Ein-/Ausgabeprofil besitzen. Funktionsmodule haben Transformations- oder Berechnungscharakter und besitzen im Gegensatz zu Modulen für Datenabstraktion keinen Zustand, der aufgehoben wird.

ganzzahlige Datentypen

`short, int, long und unsigned`-Formen in C (`INTEGER, CARDINAL` in Modula–2) zur Deklaration ganzzahliger Datenobjekte (Zahlenwerte) mit den üblichen Operationen.

getrennte unabhängige Übersetzung

Heutige Programmiersprachen und damit auch C++ und Modula-2 gestatten die separate Übersetzung verschiedener Teile eines Programmsystems. Dies erleichtert die arbeitsteilige Softwareentwicklung. Man spricht von unabhängiger Übersetzung, wenn bei separater Übersetzung keine oder nur unzureichende Querprüfungen erfolgen. Bei getrennter Übersetzung folgt die Prüfung genauso streng, als wenn das ganze Programmsystem dem Übersetzer vorgelegen hätte. Modula-2 ist nah an getrennter Übersetzung, C++ bietet einen Zwischenzustand zwischen unabhängiger und getrennter Übersetzung.

Gültigkeitsbereich

Dies ist der statische Bereich des Quelltextes, in dem eine Deklaration verwendet werden kann. Der Gültigkeitsbereich einer Deklaration reicht von der Deklaration selbst bis zum Ende der entsprechenden Programmeinheit (Unterprogramm oder Modul). Außerhalb dieses Bereichs darf keine Verwendung hingeschrieben werden.

Graphen

Allgemeinste Datenstruktur zum Ablegen/ Auffinden beliebiger netzartig strukturierter Sachverhalte. In praxi sind Graphen markiert, d.h. Knoten und Kanten gehören verschiedenen Arten an. Knotenorientierte Realisierungen legen Nachfolger– oder Vorgängerknoten in Listen ab (sequentiell oder verkettet). Charakteristische Speicherung (Adjazenzmatrix) speichert nicht die Kante, sondern nur die Information, ob eine Kante vorliegt. Kantenorientierte Realisierung behandeln Kanten als eigenständige Einheiten.

Halde

mit dem Anlegen von Speicher mittels `new` in C++ bzw. `ALLOCATE` in Modula-2 wird auf einem besonderen Speicher, Halde genannt, ein sogenanntes Haldenobjekt erzeugt. Darüber hinaus wird ein Zeigerwert darauf eingerichtet. Mit `delete` bzw. `DEALLOCATE` erfolgt die Freigabe. Anlegen bzw. Freigeben erfolgt über den Anweisungsteil eines Programms. Weitere Erläuterungen siehe → Zeiger.

Header-File

werden in C++ per Konvention zur Simulation von Modulen als Modulschittstelle eingesetzt (→ Simulation von Modulen).

Heapsort

Verbessertes Sortierverfahren nach der Strategie ”Sortieren durch Auswählen”.

Implementation-File

werden in C++ per Konvention zur Simulation von Modulen als Modulrumpf verwendet (→ Simulation von Modulen).

Information Hiding

ist ein wichtiges Prinzip bei der Softwarekonstruktion. Realisierungsdetails werden verborgen (einer Funktion, eines Bausteins einer bestimmten Art, Spezialfall, Klasse). Die Funktionalität wird über die Schnittstelle geliefert. Das → Datenabstraktionsprinzip nutzt Information Hiding für einen bestimmten Zweck.

Klassen

Abstrakte Datentypmodule, die mittels der Vererbungsbeziehung (→ Vererbungsbeziehung) miteinander in Beziehung gebracht werden können. Sie sind somit i.a. in Vererbungshierarchien eingebettet. Die Schnittstelle einer Klasse in C++ wird in den Public-, Protected- und Private-Teil unterteilt. Der Public-Teil ist für alle importierenden Bausteine sichtbar, der Protected-Teil nur für Unterklassen einer Klasse. Der Private-Teil ist nur der Klassenrealisierung (dem Rumpf der Klasse) selbst zugänglich.

Kommentare

Spezielle lexikalische Einheiten von Programmiersprachen, die vom Compiler überlesen werden. In C++ beginnen Sie mit `//` und gehen bis zum Zeilenende. Alternativ wird ein (mehrzeiliger) Kommentar in `/* */` eingeschlossen (in Modula-2 sind Kom-

mentare durch (* und *) eingerahmt). Kommentare sollten unbedingt in einem Programm verwendet werden. Sie dienen dem Festhalten von Lösungsentscheidungen und Lösungsideen. Kommentare können vor oder nach einem Programmteil stehen oder quellzeilenbegleitend sein.

Komplexität

Komplexitätsbetrachtungen untersuchen das Ergebnis der Programmentwicklung (Produktkomplexität). Die Komplexität des Entwicklungsprozesses wird i. d. R. nicht betrachtet. Meist beschränkt man sich auf die Betrachtung der Laufzeit– und der Speicherplatzkomplexität. Diese werden zur Programmlaufzeit gemessen (Monitoring) bzw. berechnet. Letzteres ist schwierig und Gegenstand der Komplexitätsuntersuchungen. Deshalb beschränkt man sich oft auf die Angabe von oberen Schranken ohne Berechnung ihrer inhärenten Konstanten (O–Notation in der Vorlesung).

Konstantendeklaration

Gibt es in C++ und Modula-2. Durch sie werden für einen compilezeitbestimmten Wert (Literal oder durch Literale gebildeter Ausdruck unter Nutzung weiterer Konstanten) ein Bezeichner eingeführt. Dies dient der Lesbarkeit und der Wartbarkeit von Programmen.

Konstrukte von Programmiersprachen

Konstrukte von Programmiersprachen sind die in einer jeweiligen Sprache eingebetteten Hilfsmittel zur Strukurierung von Programmen. Diese werden unterschieden in solche für die Ablaufkontrolle (Kontrollstrukturen), diejenigen zur Strukturierung von Daten (Datentypkonstruktoren) sowie solche zur Strukturierung der Gesamtstruktur (Modularisierungskonstrukte).

Kontrollierte Sprünge

Durch `break` können in C++ Schleifen (und nur diese) vorzeitig verlassen werden. Hierzu dient in Modula-2 die `Exit`-Anweisung, die sogar für das Verlassen beliebiger Kontrollstrukturen dient. In C++ gibt es ferner die Möglichkeit durch `continue` einen Schleifendurchlauf abzubrechen, um den nächsten Durchlauf zu beginnen.

Kontrollstrukturen

Kontrollstrukturen (→ Konstrukte von Programmiersprachen) dienen der Festlegung der Ablaufkontrolle. Hierzu zählen Anweisungssequenz, Fallunterscheidungen (bedingte Anweisung, Auswahlanweisung), Schleifen (Zählschleife, bedingte Schleifen) sowie Formen von Sprüngen. Die Kontrollstrukturen einer Programmiersprache ergeben sich aus den verschiedenen Formen für Anweisungen der Sprache.

Kurzform-Operatoren

Arithmetische Operatoren in Kurzform in C++ zur Abkürzung von binären Operationen und anschließender Zuweisung. (z.B. +=)

Kurzschluß-Auswertung

Für Boolesche Ausdrücke, die nicht vollständig ausgewertet werden, wenn der Wert des Gesamtausdrucks bereits feststeht. Oft stellt der erste Teilausdruck sicher, daß weitere überhaupt ausgewertet werden können.

Laufzeitkeller

Bei gegenseitigem Aufruf von Unterprogrammen (insbesondere bei rekursiven) ergibt sich zur Laufzeit ein kellerartig organisierter, dynamischer Datenspeicheranteil. Dieser enthält neben Formalparametern (call by value) und lokalen Variablen des Unterprogramms auch organisatorische Information für das Programmiersystem (sog. statischer und dynamischer Link). Dieser Teil des dynamischen Speicherbereichs ist deshalb nach dem Kellerprinzip organisiert, weil der Speicherbereich der zuletzt aufgerufenen Prozedur nach deren Ende als erster freigegeben werden kann.

Lexikalische Einheiten

Zusammenfassung der Syntax auf unterster Ebene von Zeichen zu sogenannten lexikalischen Einheiten (Tokens). Lexikalische Einheiten sind Bezeichner, Wortsymbole, Begrenzer, Literale und Kommentare.

Library-Files

Zusammenfassung von Header-Files vordefinierter Bausteine.

Listen

Standard-Datenstrukturen, die an vielen Stellen benötigt werden. Bei linearen Listen wird eine totale Ordnung der Listenelemente angenommen. Spezielle lineare Listen erlauben den Zugriff nur an den Enden der linearen Liste (Keller oder Stapel; Puffer oder Schlange). Lineare Listen können auf verschiedene Weise implementiert werden (sequentielle oder verkettete Realisierung). Die Verkettung kann über Zeiger oder Indizes (Cursor-Realisierung) erfolgen. Letzteres wird dann genutzt, wenn die Programmiersprache keine Zeiger besitzt. Man unterscheidet einfach verkettete Realisierung (Zeiger/

Indizes nur in eine Richtung) von mehrfach verketteter (in beide Richtungen). Bei sequentieller Realisierung wird die logische Reihenfolge (Hintereinanderreihenfolge) in einem sequentiellen Speichermedium (Feld, Datei) abgebildet. Bei der zirkulären Speicherung wird der sequentielle Speicher als geschlossen angenommen (vgl. zirkuläre Realisierung eines Puffers in der Vorlesung).

Literale

dienen dem direkten Hinschreiben eines konstanten Werts in einem Programm. Wir unterscheiden numerische Literale, Zeichen– und Zeichenkettenliterale. Numerische Literale werden in ganzzahlige Literale und reelle Literale unterschieden. Für erstere gibt es in C++ und Modula–2 oktale, dezimale und sedezimale ganzzahlige Literale.

Mengen

Für Teilmengen einer festen Trägermenge bietet Modula–2 einen Datentypkonstruktor `SET OF`. Kardinalitätseinschränkungen der Trägermenge sind überlicherweise 16, 32 o. ä. Intern wird charakteristische Speicherung benutzt, d.h. Ablage der Zugehörigkeitsinformation eines Elements zur Teilmenge. Dies führt intern zur Handhabung durch Bitvektoren. Die Mengenoperatoren werden so zu logischen Operatoren auf Bitvektoren. Für Mengen, die sich zur Laufzeit verändern und keine feste Trägermenge besitzen, sind obige Mengen nicht geeignet. Hier muß ein eigener Datentyp gebildet werden (→ abstrakter Datentyp), der über Listen, Bäume etc. realisiert wird.

Methode

Operation einer → Klasse

Modul

Ein Modul ist ein Baustein eines Softwaresystems. Er steht in Verbindung zu anderen Bausteinen. Die Exportschnittstelle gibt an, welche Dienste (Typen und Operationen) der Baustein offeriert. Die Importschnittstelle legt fest, welche Dienste anderer Bausteine zur Realisierung des Bausteins genutzt werden dürfen und müssen. Module dienen der getrennten Bearbeitung in einem Softwaresystem. Module müssen auf entsprechende Einheiten der Programmiersprache abgebildet werden (→ Modularisierungskontrukt). Module können zu Bibiliotheken zusammengefaßt werden. Sie stellen Einheiten der Wiederverwendung dar. Die Gesamtheit aller Bausteine (Module wie auch Teilsysteme) und ihrer Beziehungen ist in der Softwarearchitektur (Bauplan, Entwurfsspezifikation) festgehalten. Module sollten das Prinzip des Information Hiding beachten, d.h. daß Realisierungsdetails außerhalb des Moduls nicht sichtbar sind. Inwieweit die zugrundeliegende Programmiersprache dieses unterstützt, hängt von deren Modularisierungskonstrukten ab. Modula-2 kennt Module, die dort in Form zweier Einheiten, nämlich Schnittstelle (definition module) und Rumpf (implementation module) festgehalten werden. C++ kennt keine expliziten Modularisierungskonstrukte bis auf Klassen. Andere Bausteine müssen auf Dateiverwaltungsebene (→ Header- und → Implementation Files) gehandhabt werden.

Modula−2

Programmiersprache klassischer Bauart (imperativ, prozedural) mit Softwaretechnikeinfluß (Modularisierungskonstrukte). Modula–2 ist gut für die Ausbildung geeignet, in der Industrie und Programmentwicklung weniger verbreitet. Modula–2 wurde von N.Wirth in den 80er Jahren definiert. Auf PCs stehen Public Domain Compiler zur Verfügung. Modula–3, der Nachfolger von Modula–2, besitzt auch Konstrukte für die Objektorientierung.

Modularisierungskonstrukte

Konstrukte (→ Konstrukte von Programmiersprachen) zur Festlegung der Grobstruktur eines Programms. Hierzu zählen Prozeduren, Module, Klassen sowie die Möglichkeit der Bildung größerer Einheiten mit ihnen (Teilsysteme).

offene Felder

für Formalparameter von Unterprogrammen in Modula–2. Diese Felder müssen eindimensional sein. Innerhalb der Prozedur beginnt der Index des Aktualparameters bei 0. Die obere Grenze ist nicht festgelegt. Offene Felder dienen unter obigen Einschränkungen zur Formulierung von Unterprogrammen, die unabhängig von den Indexgrenzen und damit der Komponentenzahl des Feldes sind.

Parameterzuordnung (s−Mechanismus)

Mechanismus zur Zuordnung (Übergabe) von Aktualparametern zu Formalparametern bei Unterprogrammen. Die Üblichen sind call by value (Aufruf über den Wert) und call by reference (Aufruf über die Adresse). Im ersten Fall fungiert der Formalparameter als lokale Variable. Der Wert des Aktualparameters wird kopiert. Solche Parameter dienen zur Eingabe von Werten in das Unterprogramm (Eingangsparameter). Im zweiten Fall wird der Aktualparameter (seine

Adresse) zur Bindung herangezogen (direkte Veränderung des Aktualparameters). Diese zweite Art dient der Handhabung von Ein–/Ausgabeparametern (Transienten) und Ausgabeparametern.

Pascal

Ausbildungssprache die etwa 1970 von N. Wirth definiert wurde. Pascal besitzt keine Hilfsmittel für das Programmieren im Großen. Die Kontrollstrukturen und Datentypkonstruktoren von Pascal sind mittlerweile Standard in klassischen Programmiersprachen. Pascaldialekte besitzen zahlreiche Erweiterungen in allerdings nicht standardisierter Form.

Portabilität

nennt man die Eigenschaft eines Programmsystems, unabhänig von der konkreten Plattform (Hardware, Programmiersystem, Dateiverwaltungssystem, Ein–/Ausgabesystem etc. und deren Versionen) zu sein.

Prinzipien → Strategie

Programmiersprachenkonstrukte

→Konstrukte von Programmiersprachen

Programmieren im Großen

Systematische Vorgehensweise bei der Gestaltung großer Programmsysteme. Hierzu ist ein Bauplan (eine Architektur) zu erstellen, die Export– und Importschnittstellen der Bausteine sind festzulegen. Die Realisierung der Bausteine ist Gegenstand des Programmierens im Kleinen. In der Vorlesung taucht Programmieren im Großen nur andeutungsweise auf. Die Schnittstelle festgelegter Bausteine erfolgt normalerweise allein auf syntaktischer Ebene, d.h. der Festlegung des Typs und der Reihenfolge der Formalparameter von Operationen der Bausteine.

Programmieren im Kleinen

Systematische Vorgehensweise bei der Ausgestaltung kleiner Programme oder der Ausgestaltung von Programmbausteinen bzw. die Ausgestaltung selbst. Somit faßt der Begriff die methodische Vorgehensweise als auch die Detailrealisierung solcher Bausteine zusammen.

Prototyp

Spezielle C++-Terminologie für Funktionskopf.

Quicksort

Verbessertes Sortierverfahren nach der Strategie des ”Sortierens durch Vertauschen”. Es wird häufig verwendet. Allerdings ist Quicksort im schlechtesten Fall von quadratischem Aufwand.

Records → Verbunde

reelle Datentypen

(`float`, `double` in C, `FLOAT` in Modula–2,) für numerisch reelle Datentypen (Zahlenwerte) mit den üblichen arithmetischen Operationen.

Referenzen

Eine spezielle Form von Zeigern in C++. Solche Zeiger müssen bei der Deklaration gesetzt werden und werden nicht verändert. Bei Zugriffen muß nicht dereferenziert werden. Auf diese Weise können Zugriffswege auf Datenobjekte über Adressen eingerichtet werden.

Reihungen → Felder

Robustheit

nennt man die Eigenschaft eines Programmsystems, wenn falsche Bedienereingaben vollständig abgeprüft werden und somit nicht zur Programmbeendigung oder sogar zum Programmabsturz führen.

Rumpf eines Moduls

Der Rumpf eines Moduls (→ Modul) realisiert die Exportschnittstelle unter Nutzung der Dienste (Ressourcen) der Importschnittstelle. Hierzu werden geeignete Datenstrukturen (Deklarationsteil) und Ablaufstrukturen (Anweisungsteil) festgelegt. Ggfl. werden lokale Prozeduren im Deklarationsteil definiert, wenn diese nicht groß sind oder nicht anderweitig verwendbar sein sollen. In letzterem Falle werden sie selbst zu Modulen.

Schleifen

dienen zur wiederholten Ausführung einer Anweisungsfolge (des Schleifenrumpfes). Wir unterscheiden Zählschleifen und bedingte Schleifen. Letztere werden wiederum unterschieden in while–Schleifen (Fortsetzungsbedingung, die vor dem Schleifenrumpf ausgewertet wird) und until–Schleifen (Abbruchbedingung, die nach dem Schleifenrumpf ausgewertet wird). Bei bedingten Schleifen ist unbedingt auf Termination zu achten. Hierzu wird in den Terminationsüberlegungen eine Monotonieüberlegung angestellt und eine Schranke gefunden, die erreicht werden muß.

Schnittstelle eines Moduls

Die Schnittstelle eines Moduls (→ Modul) besteht aus der Export- und der Importschnittstelle. Manchmal wird auch nur die erstere als Schnittstelle bezeichnet.

schrittweise Verfeinerung

nennt man die Programmentwicklungs–Methode, über Pseudoanweisungen (abstrakte Anweisungen) einen Anweisungsteil schrittweise zu verfeinern, bis man auf der Ebene der Programmiersprache angelangt ist. Die Pseudoanweisungen bleiben gegebenenfalls als Kommentare zurück.

Seiteneffekt(freiheit)

Ausdrücke sollten einen Wert zurückliefern und bei ihrer Auswertung keine Seiteneffekte erzeugen. Dies bedeutet z.B., daß in C++ keine ++–Operatoren o. ä. in Ausdrücken verwendet werden sollten. Ebenso sollten Funktionen keine Seiteneffekte erzeugen (z.B. Verändern globaler Variablen).

selbstdefinierte Datentypen → Typ

Sichtbarkeitsbereich

ist ein Teilbereich des Gültigkeitsbereichs, in dem eine Deklaration angesprochen werden kann. Der Sichtbarkeitsbereich ist ein Teilbereich, da Verdeckung von Variablen, Unterprogrammen (Überladung in C++) auftreten kann. In C++ kann eine verdeckte Deklaration durch den Scope-Operator `(::)` angesprochen werden, bei überladenen Unterprogrammen sucht der Compiler das richtige heraus.

Simulation von Modulen in C++

Die Schnittstelle eines Moduls wird in C++ in einem Header-File zusammengefaßt, die Realisierung des Modulrumpfes in einem Implementation-File. Dies ist zunächst ähnlich zu Modula-2 (Definition und Implementation Module). Zwischen beiden Dateien erfolgt in C++ aber keine Überprüfung auf Konsistenz. Bei Importen (Einbeziehen anderer Header-Files über `#ifndef` und `#define` erfolgt in C++ kein Import auf der Ebene einzelner Ressourcen eines Moduls. Bei sehr disziplinierter Verwendung von Header-Files, Kommentierung und disziplinierter Programmierung von Modulen unter Bezugnahme auf die Header-Files anderer Module können in C++ nicht vorhandene Modularisierungskonstrukte simuliert werden. Dies wird in der Vorlesung gezeigt (Funktionsmodule, Datenobjekt- und Datentypmodule sowie ihr Import für weitere Module).

skalare Datentypen → Typ

Sortieren

nennt man das Anordnen von Datenelementen nach einem Teil ihrer Information (Schlüssel oder Primärschlüssel). Die Werte des Schlüssels müssen total geordnet sein. Dies ist für ganzzahlige Werte der Fall bzw. für Zeichenwerte. Bei zusammengesetzten Schlüsseln wird lexikographische Ordnung herangezogen. Sortieren wird unterschieden in batchartiges oder inkrementelles, internes oder externes. In der Vorlesung haben wir direkte Sortierverfahren (einfache Verfahren) kennengelernt, z.B. Bubblesort. Quicksort wurde als Beispiel eines verbesserten Verfahrens behandelt (im Mittel O(n logn).

stepwise refinement

→ schrittweise Verfeinerung

Strategien (Prinzipien für Algorithmen)

Strategien fassen bestimmte Algorithmen zu einer Klasse zusammen. Alle Repräsentanten der Klasse folgen der gleichen Strategie. Beispielsweise gibt es für ein Sortierverfahen 4 Strategien (Einfügen, Austauschen, Auswählen, Mischen). Analoges gilt für Suchverfahren. Durch Festlegung von Details wird aus einer Strategie ein konkretes Verfahren. In der Vorlesung haben wir z.B. zwei Verfahren (Bubblesort, Quicksort) kennengelernt, die der Strategie "Sortieren durch Vertauschen" folgen. Andere Klasseneinteilungen für Sortierverfahren sind intern oder extern, direkte oder verbesserte Verfahren, stabile oder instabile Verfahren, batchartige oder inkrementelle Verfahren.

Struktogramm

Graphische Darstellung für Programme. Dabei wird nur die Ablaufkontrolle dargestellt. Für die üblichen Kontrollstrukturen gibt es entsprechende graphische Elemente (Sequenz, bedingte Anweisung, Auswahlanweisung, Schleifen, Prozeduren).
→ AnhangII

Strukturen → Verbunde

Suchen

nennt man ein Programm zum Auffinden von Datenelementen, wenn deren Schlüssel bekannt ist. Sortieren ist eine Voraussetzung für effizientes Suchen, um erschöpfende Suche zu vermeiden. Bei binärer Suche wird fortgesetzte Intervallschachtelung angewendet. Binäre (ausgeglichene) Suchbäume sind eine Technik zur einfachen Realisierung binärer Suche auf i. d. R. veränderlichem Datenbestand.

Syntaxnotation

Formale Festlegung der Syntax einer Programmiersprache. Hierzu bedarf es einer formalen Sprache zur Festlegung dieser Syntax. In der Vorlesung wird lediglich die kontextfreie Syntax formal durch EBNF–Regeln

oder durch Syntaxdiagramme definiert. Kontextsensitive Syntax wird bei Programmiersprachendefinitionen in der Regel durch Prosa erläutert.

Test

Experimentelle Überprüfung eines Programms. Test wird gegliedert in Testdatengewinnung, Testvorbereitung, Testdurchführung und Testauswertung. Der Modultest dient der Überprüfung einzelner Bausteine, der Integrationstest der Überprüfung von Gruppen von Bausteinen, der Systemtest der des gesamten Programms. In letzterem Fall werden auch Funktions- und Leistungsüberprüfung betrachtet. Gewisse Schritte des Tests sind automatisierbar oder teilweise automatisierbar (Testdatengewinnung, Erzeugen Teststummel und Testtreiber, Durchführung von Regressionstest). Durch Testen kann nur die Anwesenheit, aber nicht die Abwesenheit von Fehlern festgestellt werden (Dijkstra)

Trace

→ s. Ablaufverfolgung

Typ

Datentypen unterscheidet man in vordefinierte oder selbstdefinierte. Eine andere Einteilung unterscheidet einfache und zusammengesetzte. Programmiersprachen besitzen in der Regel sowohl einfache als auch zusammengesetzte vordefinierte Datentypen, die meisten der vordefinierten Datentypen sind einfache Datentypen (skalare Datentypen). Ein Typ faßt die Struktur einer Klasse von Objekten zusammen (Klasse hier nicht im objektorientierten Sinne): Struktur, Wertebereich, sowie deren Operationen. Operationen sind oft implizit, z.B. `+` für `int`.

Typgleichheit/Typäquivalenz

Hierunter versteht man die Regeln einer Programmiersprache, die festlegen, wann zwei Objekte vom Typ her als gleich zu betrachten sind. Bei Programmiersprachen mit Strukturäquivalenz wird die Struktur von Typdefinitionen untersucht. Modula–2 führt keine Strukturäquivalenz ein. Jede Typdefinition (Strukturfestlegung eines Typs) führt einen neuen Typ ein! Mit einer Synonymdeklaration können verschiedene Datentypen eingeführt werden, die als äquivalent betrachtet werden.

Überladung von Prozeduren

Gibt es nur in C++. Damit sind verschiedene Unterprogramme mit gleichem Namen definierbar, die sich anhand des Parametertyp-Profils (Anzahl, Reihenfolge, Typ der Formalparameter) unterscheiden müssen. In der kontextsensitiven Syntaxanalyse wird das passende Unterprogramm herausgesucht, das dabei eindeutig definierbar sein muß.

Unterbereichstypen

In Modula–2 für skalare Datentypen möglich zur Festlegung eines laufzeitabhängigen Bereichs. Unterbereichstypen passen nicht in das sonstige Typkonzept, da sie keine strukturellen Festlegungen treffen, sondern nur Laufzeiteinschränkungen wiedergeben.

Unterprogramme

Hilfsmittel zur Grobstrukturierung von Programmen (→ Modularisierungskonstrukte). Unterprogramme bestehen aus einem Unterprogrammkopf (Unterprogrammschnittstelle mit Formalparameterliste). Unterprogramme liefern einen Wert (Funktion) oder haben den Charakter einer zusammengesetzten Anweisung (Prozeduren, Verfahrensprozeduren). Im Unterprogramm–Rumpf wird die Berechnung ausformuliert. Hierzu können Deklarationen eingeführt werden und der Anweisungsteil kann beliebig strukturiert sein. Funktionen bzw. Prozeduren sind in C++ als auch in Modula-2 durch die Form des Unterprogrammkopfes unterscheidbar.

variante Verbunde

Variante Verbunde sind Verbunde (→ Verbunde) mit einem gemeinsamen Teil und unterschiedlichen Varianten. Variantenauswahl wird durch die Diskriminante, meist eine Variable eines Aufzählungstyps, gesteuert. Die verschiedenen Varianten können unterschiedlich strukturiert sein und somit auch unterschiedlich lang sein. Die Diskriminante ist Teil des Verbunds. Der Compiler legt soviel Platz an, wie der längste Variantenteil erfordert.

Verbunde (Strukturen, Records)

Zusammenfassung verschiedenartiger Datenobjekte zu einem Ganzen. Zugriff über Selektornamen. Der Selektorpfad steht zur Programmerstellungszeit fest. Man unterscheidet einfache Verbunde von varianten Verbunden. Bei varianten Verbunden folgen einem gemeinsamen Teil unterschiedliche Varianten. Ein konkretes Datenobjekt kann zu einem bestimmten Zeitpunkt der Programmausführung nur einer Variante angehören.

Vererbungsbeziehung

Aus Klassen können Spezialisierungen gewonnen werden. Diese Beziehung heißt Spezialisierung oder Vererbung, da die Unterklasse von der Oberklasse deren Eigen-

schaften erbt. Die umgekehrte Beziehung heißt Generalisierung. Klassen werden in Vererbungshierarchien angeordnet, um so die Ähnlichkeit zwischen verschiedenen Klassen ausdrücken zu können. Es ist i.a. schwierig, saubere Vererbungshierarchien aufzubauen. Falls eine Klasse nur eine Oberklasse haben darf, spricht man von Einfachvererbung, ansonsten von Mehrfachvererbung. Objektorientierte Softwarekonstruktion heißt, daß ein Softwaresystem oder ein Teil hiervon nur aus Vererbungshierarchien aufgebaut wird.

vordefinierte Datentypen → Typ

White-Box-Test

Modultest unter Betrachtung der internen Realisierung eines Bausteins. Üblich sind sogen. Überdeckungsverfahren. In der Vorlesung haben wir die Flußgraphenmethode kennengelernt, welche die sämtlich möglichen Pfade der Ablaufkontrolle betrachtet. Zur Reduktion der Testfälle werden dabei mehrmalige Schleifendurchläufe zu einem Muster zusammengefaßt. Die entsprechenden möglichen Ablaufmuster werden einzeln durch Testfälle betrachtet.

wohlstrukturierte Programme

wohlstrukturierte Programme besitzen keine explizite Sprunganweisung. Sie sind ausschließlich mittels Sequenz, Fallunterscheidung (`if, switch`), Iteration (`for, while`) gebildet. In der sogenannten goto–Kontroverse wurde das Für und das Wider von Sprunganweisungen ausführlich und zum Teil ideologisch diskutiert. Kontrollierte Sprünge sind Sprunganweisungen, die keine weiteren, als die gegebenen Sprungmarken der Kontrollstrukturen einführen. Verwendet man diese, so spricht man auch von `EXIT`–wohlstrukturierten Programmen.

Wortsymbole

Bezeichner mit vordefinierter Semantik in einer Programmiersprache. Meist dürfen sie, in jedem Falle sollten sie nicht für selbstdefinierte Bezeichner verwendet werden (Beispiele `if, do, const` in C++; `IF, TYPE` in Modula-2).

Zählschleifen

Eine Form von Schleifen (→ Kontrollstrukturen, → Anweisungen) zur wiederholten Ausführung eines Programmteils, des Rumpfs der Schleife, der eine Anweisungsfolge darstellt. Zählschleifen haben allgemein kein Terminationsproblem. Im Schleifenkopf sind Anfangswerte, Endwerte und ggfl. Schrittweite angegeben. Alle Werte können laufzeitabhängig sein.

Zeichenketten

werden als Felder mit Komponenten des Typs `char` in C++ bzw. `Character` in Modula–2 betrachtet. Zeichenketten können durch Zeichenkettenliterale dargestellt sein bzw. durch Zeichenkettenausdrücke mithilfe des Konkatenationsoperators gebildet werden.

Zeichentypen

`char` in C, `CHAR` in Modula–2 mit den Zeichen des jeweiligen, zugrundeliegenden Alphabets.

Zeiger

Mithilfe von Zeigertypen (→ Datentypkonstruktoren) können beliebige vernetzte Datentypen deklariert werden. Diese dienen einerseits dazu, beliebige Relationen zwischen Datenobjekten festlegen zu können. Zum anderen können rekursive Datentypen damit definiert werden (z.B. Bäume). Zeiger sind in Modula–2 typisiert, d.h. sie können nur auf Objekte eines bestimmten Typs verweisen. Zeiger werden zur Laufzeit neu eingerichtet bei der Erzeugung von Haldenobjekten (Allokation, `new` in Modula-2) oder durch Setzen in einer Wertzuweisung. Durch Löschen von Haldenobjekten können hängende Zeiger entstehen (dangling references). Mit Haldenobjekten, Zeigern auf Haldenobjekten und Verweisen zwischen Haldenobjekten über Zeiger können beliebig komplexe Datenstrukturen gehandhabt werden. Solche Strukturen sollten möglichst modullokal gehalten werden. Neben hängenden Zeigern und unübersichtlichen Datenstrukturen ergeben sich Probleme mit verschiedenen Zugriffspfaden (Aliasing) sowie nicht mehr zugreifbaren Haldenobjekten.

zusammengesetzte Datentypen → Typ