

Algorithmen und Datenstrukturen

- Dynamische Datenobjekte
 - Pointer/Zeiger, Verkettete Liste
- Eigene Typdefinitionen

Zeigeroperatoren & und *

- Ein Zeiger ist die Speicheradresse irgendeines Objektes.
- Eine Zeigervariable ist eine Variable, die gemäß Deklaration einen Zeiger auf ein Objekt des vereinbarten Typs enthält.
- Mit dem **Adressoperator &** kann man die Adresse einer Speicherzelle ermitteln. Der Adressoperator kann auf Variablen und Arrayelemente angewendet werden, nicht auf Ausdrücke (sie haben keinen Speicherplatz).
- Da ein Zeiger die Adresse eines Objektes enthält, kann auf das Objekt auf zwei Arten zugegriffen werden:
 - Direkt über den Namen
 - Indirekt über den Zeiger.
- Um indirekt zuzugreifen, wird der Operator * zum **Dereferenzieren** verwendet.

Zeigeroperatoren & und *

- Wird eine Zeigervariable deklariert, so ist sie nicht automatisch initialisiert; sie zeigt irgendwo hin.
- Will man sie auf einen definierten Wert setzen, aber noch nicht auf ein bestimmtes Objekt, so kann der spezielle Zeigerwert **NULL** oder (neu) **nullptr** verwendet werden.
- **NULL** (**nullptr**) wird intern als **0L** dargestellt; man sollte der besseren Lesbarkeit wegen aber den Namen **NULL** bzw. **nullptr** verwenden.
- Die Definition von **NULL** ist im Header `<cstdlib>` enthalten.
- In den folgenden Folien wird **NULL** (statt **nullptr**) angegeben.

Dirk Seeber , Informatik , Teil 5

3

Zeigeroperatoren & und *

- In Ausdrücken, wie $y = *p + 1$; haben die Operatoren ***** und **&** **höheren** Rang als arithmetische Operatoren.
- D.h. **zuerst** wird oben das Objekt ermittelt (**dereferenziert**) dann wird der Wert um 1 erhöht.
- Achtung: $y = *(p+1)$ möglich, hat aber andere Bedeutung!!
- Verweise sind auch links in Zuweisungen möglich.

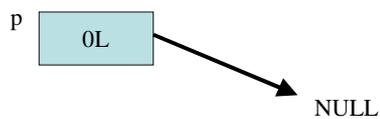
Dirk Seeber , Informatik , Teil 5

4

Zeiger

- Beispiele für Zugriffe:

C++-Fragment	Bedeutung
<pre>int *p = NULL;</pre>	p ist eine Zeigervariable auf ein int-Objekt. Da keine Initialisierung vorgenommen ist, ist der Inhalt unbestimmt.



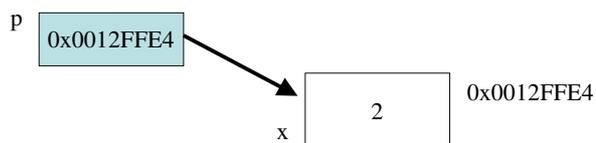
Dirk Seeber , Informatik , Teil 5

5

Zeiger

- Beispiele für Zugriffe:

C++-Fragment	Bedeutung
<pre>int *p = NULL; int x = 2; p = &x;</pre>	p wird die Adresse von x zugewiesen. Man sagt p zeigt auf x.



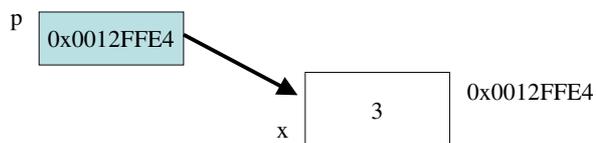
Dirk Seeber , Informatik , Teil 5

6

Zeiger

- Beispiele für Zugriffe:

C++-Fragment	Bedeutung
<pre>int *p = NULL; int x = 2; p = &x; x = *p + 1;</pre>	p zeigt auf x. Zuerst wird dereferenziert, d.h. der Wert von der Speicherstelle, auf die p zeigt wird ermittelt (x==2), dann der Wert erhöht (3) und der Variablen x zugewiesen.



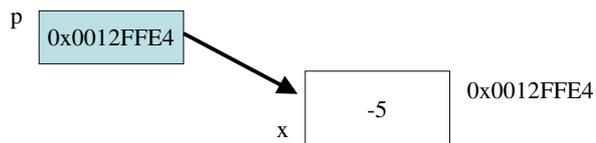
Dirk Seeber , Informatik , Teil 5

7

Zeiger

- Beispiele für Zugriffe:

C++-Fragment	Bedeutung
<pre>int *p = NULL; int x = 2; p = &x; *p = -5;</pre>	p zeigt auf x. Zuerst wird dereferenziert, d.h. der Wert von der Speicherstelle, auf die p zeigt wird ermittelt (x==2), dann wird der Wert dieser Speicherstelle auf -5 gesetzt.



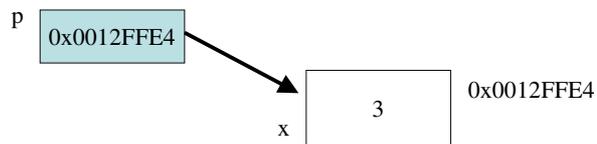
Dirk Seeber , Informatik , Teil 5

8

Zeiger

- Beispiele für Zugriffe:

C++-Fragment	Bedeutung
<pre>int *p = NULL; int x = 2; p = &x; *p += 1;</pre>	p zeigt auf x. Zuerst wird dereferenziert, d.h. der Wert von der Speicherstelle, auf die p zeigt wird ermittelt (x==2), dann wird der Wert dieser Speicherstelle um 1 erhöht.



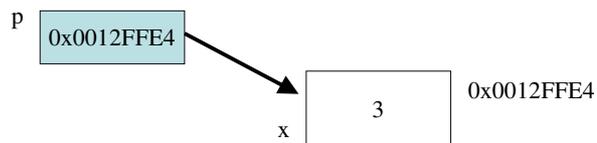
Dirk Seeber , Informatik , Teil 5

9

Zeiger

- Beispiele für Zugriffe:

C++-Fragment	Bedeutung
<pre>int *p = NULL; int x = 2; p = &x; (*p)++;</pre>	p zeigt auf x. Zuerst wird dereferenziert, d.h. der Wert von der Speicherstelle, auf die p zeigt wird ermittelt (x==2), dann wird der Wert dieser Speicherstelle um 1 erhöht. Die Klammern sind wegen Rang erforderlich.



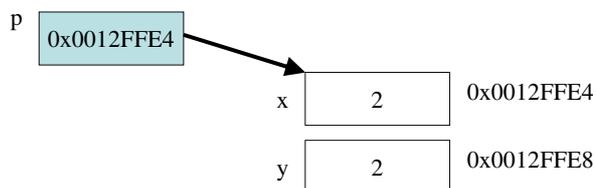
Dirk Seeber , Informatik , Teil 5

10

Zeiger

- Beispiele für Zugriffe:

C++-Fragment	Bedeutung
<pre>int *p = NULL; int x = 2, y; p = &x; y = *p;</pre>	p zeigt auf x. Durch <code>y = *p</code> wird derselbe Effekt erzielt wie durch <code>y = x</code> .



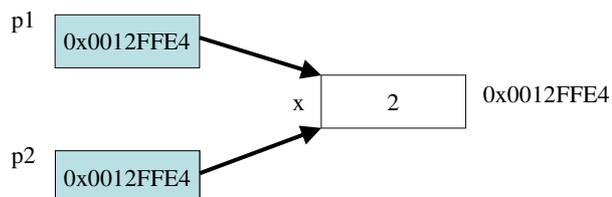
Dirk Seeber, Informatik, Teil 5

11

Zeiger

- Beispiele für Zugriffe:

C++-Fragment	Bedeutung
<pre>int *p1 = NULL; int *p2 = NULL; int x = 2; p1 = &x; p2 = p1;</pre>	p1 zeigt auf x. Der Variablen <code>p2</code> wird der Wert der Variablen <code>p2</code> zugewiesen. Dadurch zeigt <code>p2</code> auf dasselbe Objekt wie <code>p1</code> .



Dirk Seeber, Informatik, Teil 5

12

Dynamische Datenobjekte

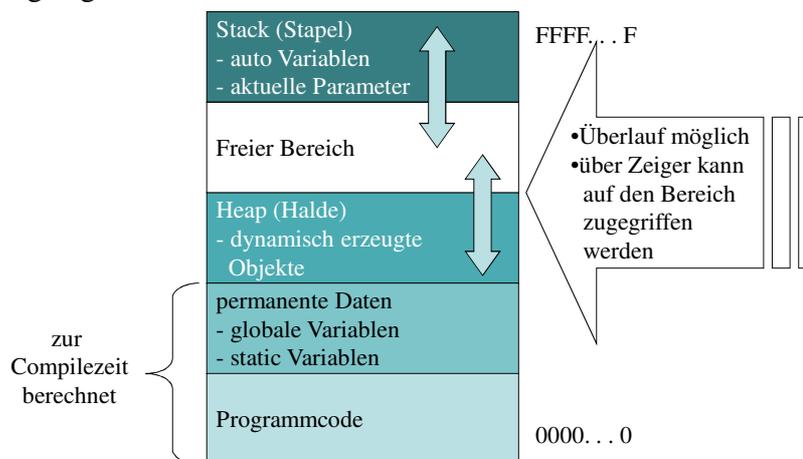
- Die bisher behandelten Datentypen waren statisch – der **Compiler** konnte den Speicherplatzbedarf ermitteln.
- In vielen Anwendungen ist der erforderliche Platzbedarf aber erst zur Laufzeit des Programms bekannt; z.B. wenn der Platz von der Anzahl der Benutzereingaben abhängt.
- Eine Lösung ist, ein sehr großes Array von Datenobjekten zu deklarieren und dort die Werte zu speichern.
- Besser wäre es man könnte Speicher zur Laufzeit, bei Bedarf anfordern (allokieren).

Dirk Seeber , Informatik , Teil 5

13

Dynamische Datenobjekte

- Ein Programm wird vom Betriebssystem (je nach Betriebssystem unterschiedlich) im Speicher etwa wie folgt abgelegt.



Dirk Seeber , Informatik , Teil 5

14

Dynamische Datenobjekte

- Der Zugriff auf Daten im Stack und den Bereich permanente Daten erfolgt über Namen (der Variablen).
- Der Zugriff auf Elemente des Heap erfolgt über Zeiger.

Erzeugung von dynamischen Datenobjekten

- Um ein **neues Element** im Heap ablegen zu können, verwendet man den **Operator new**.
- Der erforderliche Speicherplatz wird durch `new` in Abhängigkeit des Typs automatisch ermittelt.
- Beispiele für `new` Operator nächste Seite

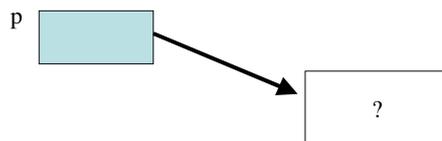
Dirk Seeber , Informatik , Teil 5

15

Dynamische Datenobjekte

- Beispiele für `new` Operator:

C++-Fragment	Bedeutung
<pre>int *p = NULL; p = new int;</pre>	Ein neues Element wird auf dem Heap angelegt. Der Zugriff erfolgt über den Zeiger p. Der Wert ist undefiniert. Die Größe ist <code>sizeof(int)</code> .



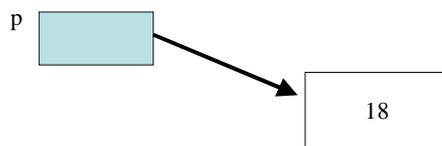
Dirk Seeber , Informatik , Teil 5

16

Dynamische Datenobjekte

- Beispiele für `new` Operator:

C++-Fragment	Bedeutung
<pre>int *p = NULL; p = new int; *p = 18;</pre>	Ein neues Element wird auf dem Heap angelegt. Der Zugriff erfolgt über den Zeiger p. Der Wert 18 wird zugewiesen. Die Größe ist <code>sizeof(int)</code> .



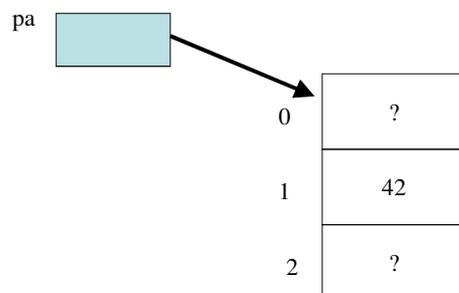
Dirk Seeber , Informatik , Teil 5

17

Dynamische Datenobjekte

- Beispiele für `new` Operator:

C++-Fragment	Bedeutung
<pre>int *pa = NULL; pa = new int[3]; pa[1] = 42;</pre>	Anlegen von 3 Int-Objekten. Der Zugriff wie auf Arrays über Zeiger und Index. Der Wert 42 wird dem 2. Element zugewiesen. Die Größe ist <code>3 * sizeof(int)</code> .



Dirk Seeber , Informatik , Teil 5

18

Erzeugung von dynamischen Datenobjekten

- Beispiel:
Ein Programm, in dem ein **Vektor von Zeigern** definiert ist, bei dem jedes Vektorelement auf ein double-Objekt zeigt.

```
#include <iostream>
int main()
{
    double *pd[10]; // Vektor von Zeigern auf double;
    for (int i=0; i<10; i++)
    {
        cout << "Double: ";
        pd[i] = new double; // Komponente i zeigt auf
                           // neues Element im Heap
        cin >> *pd[i];      // Wertzuweisung an neues
                           // Element über Zeiger
    }
    for (int i=9; i>=0; i-- )
    {
        cout << *pd[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Dirk Seeber , Informatik , Teil 5

19

Erzeugung von dynamischen Datenobjekten

- Beispiel:
Ein Programm mit einem Zeiger auf ein Array von double-Werten.

```
#include <iostream>
int main()
{
    double *vd = new double[10]; // Zeiger auf Vektor
    for (int i =0 ; i < 10; i++)
    {
        cout << "Double: ";
        cin >> vd[i]; // Wertzuweisung an neues
                     // Element über Zeiger
    }
    for (int i = 9; i >= 0; i-- )
    {
        cout << vd[i] << " ";
    }
    cout << endl;
    return 0;
}
```

Dirk Seeber , Informatik , Teil 5

20

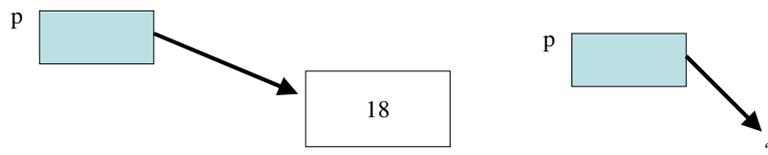
Freigabe von dynamischen Datenobjekten

- Der Operator `delete` muss verwendet werden, um ein vorher mit `new` erzeugtes Objekt wieder freizugeben, damit der Speicherplatz erneut allokiert werden kann.
- Ein Array von Datenobjekten wird mit `delete [] pa;` freigegeben.

Dynamische Datenobjekte

- Beispiele für `delete` Operator:

C++-Fragment	Bedeutung
<pre>int *p = NULL; p = new int; *p = 18;</pre>	Ein neues Element wird auf dem Heap angelegt. Der Zugriff erfolgt über den Zeiger p. Der Wert 18 wird zugewiesen.
<pre>int *p = NULL; p = new int; *p = 18; delete p;</pre>	Freigabe, auf das Element kann man nicht mehr zugreifen.



Freigabe von dynamischen Datenobjekten

- Einige Besonderheiten sind bzgl. `delete` zu beachten:
 - Dynamisch allozierter Speicher, auf den kein Zeiger mehr zeigt, ist **verloren!**
 - C++ **erkennt nicht**, ob ein Speicher noch benutzt wird (wegen der Laufzeiteffizienz).
[dies ist in Java anders: die JVM überwacht den Heap und führt automatische ein Garbage Collection durch].
 - `delete` darf ausschließlich auf Objekte angewendet werden, die mit `new` erzeugt wurden.
 - `delete` auf einen NULL Zeiger ist wirkungslos.
 - mit `new` erzeugte Datenobjekte unterliegen nicht den Gültigkeitsregeln für Variablen.
Sie existieren solange, bis mit `delete` gelöscht werden.

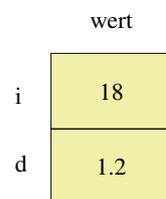
Rekursive dynamischen Strukturen

- Zeiger sind nicht nur auf einfache Objekte möglich.
- Ein Zeiger kann auch auf eine Struktur zeigen.
- Da ein Element der Struktur selbst wieder ein Zeiger sein kann, entsteht eine rekursive Datenstruktur.
- Wenn ein Zeiger auf eine Struktur zeigt, kann eine Komponente selektiert werden durch Verwendung des Selektion- und des Dereferenzierungsoperators.

Dynamische Datenobjekte

- Beispiele für dynamische Strukturen:

C-Fragment	Bedeutung
<pre>struct tupel { int i; double d; } wert; wert.i = 18; wert.d = 1.2;</pre>	Definition einer Variable wert, die eine Struktur von einem int und einem double ist.



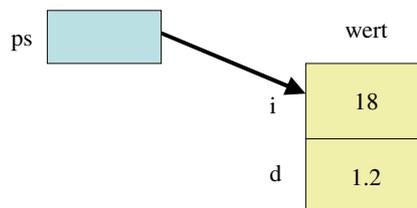
Dirk Seeber , Informatik , Teil 5

25

Dynamische Datenobjekte

- Beispiele für dynamische Strukturen:

C-Fragment	Bedeutung
<pre>struct tupel *ps; ps = &wert;</pre>	Zeiger auf Struktur tupel; ps zeigt auf Variable wert.



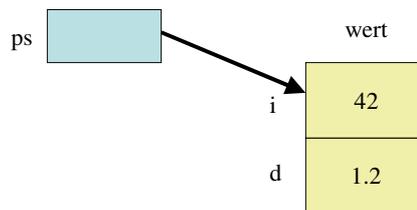
Dirk Seeber , Informatik , Teil 5

26

Dynamische Datenobjekte

- Beispiele für dynamische Strukturen:

C-Fragment	Bedeutung
<pre>(*ps).i = 42; cout << ps->i;</pre>	Komponente i der Struktur selektieren, auf die ps zeigt und den Wert 42 zuweisen. Kurzschreibweise für (*ps).i ist: ps->i



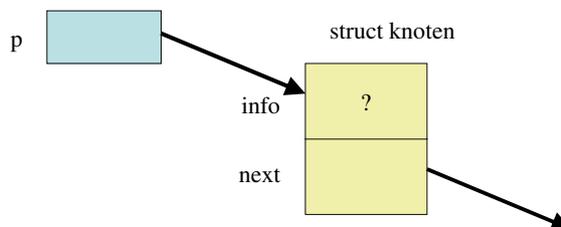
Dirk Seeber, Informatik, Teil 5

27

Dynamische Datenobjekte

- Beispiele für rekursive dynamische Strukturen:

C++-Fragment	Bedeutung
<pre>struct knoten { int info; struct knoten *next; }; knoten *p = new knoten;</pre>	Definition einer Struktur, die auf sich selbst zeigt (next-pointer). Neuer Zeiger p auf Struktur knoten.



Dirk Seeber, Informatik, Teil 5

28

Dynamische Datenobjekte

- Hier ist der Rang der Operatoren wichtig:
 - `++ps->i;`
Entspricht `++(ps->i)` also wird die Komponente `i` inkrementiert.
 - `(++ps)->i;`
Inkrementiert `ps`, zeigt also auf anderes Objekt, dann wird die Komponente `i` selektiert!

Zusammenfassung: Dynamische Datenobjekte

- Um ein **neues Element** im Heap ablegen zu können, verwendet man den **Operator new** (in C verwendet man `malloc`).
- Der erforderliche Speicherplatz wird durch `new` in Abhängigkeit des Typs automatisch ermittelt.
- Beispiele:

```
int *zeiger1;
zeiger1 = new int;
zeiger1 = NULL;

double *zeiger2 = NULL;
zeiger2 = new double[10];
```

Zusammenfassung: Dynamische Datenobjekte

- Ein Zeiger kann auch auf eine Struktur zeigen.
- Da ein Element der Struktur selbst wieder ein Zeiger sein kann, entsteht eine rekursive Datenstruktur.
- Beispiel:

```
struct student
{
    string nachname;
    string vorname;
    long matrikelnummer;
    student *next;
};

student *zeigerAufListe = new student;
zeigerAufListe->nachname = "Seeber\0";
zeigerAufListe->vorname = "Dirk\0";
(*zeigerAufListe).matrikelnummer = 123456;
zeigerAufListe->next = NULL;
```

Dirk Seeber , Informatik , Teil 5

31

Zusammenfassung: Dynamische Datenobjekte

- Der Operator `delete` (in C verwendet man `free`) kann verwendet werden, um ein vorher mit `new` erzeugtes Objekt wieder freizugeben, **damit der Speicherplatz erneut allokiert werden kann.**
- Beispiele:

```
int *zeiger1;
zeiger1 = new int;
delete zeiger1;

double *zeiger2 = NULL;
zeiger2 = new double[10];
delete [] zeiger2;
```

Dirk Seeber , Informatik , Teil 5

32

Zusammenfassung: Dynamische Datenobjekte

- Beispiel (Fortsetzung):

```
struct student
{
    string nachname;
    string vorname;
    long   matrikelnummer;
    student *next;
};

student *zeigerAufListe = new student;
zeigerAufListe->nachname = "Seeber\0";
zeigerAufListe->vorname = "Dirk\0";
zeigerAufListe->matrikelnummer = 123456;
zeigerAufListe->next = NULL;

delete zeigerAufListe;
```

Dirk Seeber , Informatik , Teil 5

33

Verkettete Listen

- Als erstes Anwendungsbeispiel werden **verkettete Listen** demonstriert.
- Eine verkettete Liste ist eine rekursive dynamische Struktur, bei dem ein Strukturelement auf ein Datenobjekt des gleichen Typs zeigt.
- Das heißt ein Knoten hat die Form:

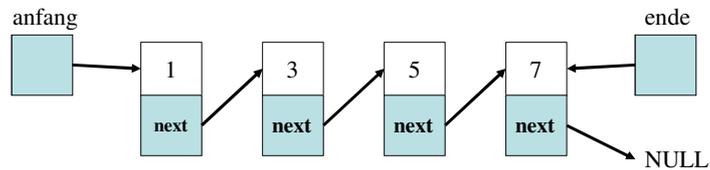
```
struct knoten
{
    int info;
    knoten *next;
};
```

Dirk Seeber , Informatik , Teil 5

34

Verkettete Listen

- Damit lassen sich dann Listenelemente verketteten:



Dirk Seeber , Informatik , Teil 5

35

Verkettete Listen - Operationen

- Folgende Operationen werden nun auf der verketteten Liste implementiert:
 - Ein **neues Element** mit info-Wert `neuWert` soll immer am **Ende** der Liste eingefügt werden.

```
void einfuegen( int neuWert, knoten **anfang,
               knoten **ende );
```
 - Die Liste wird vom Anfang an bis zum Ende durchlaufen und der info-Teil jeden Knotens wird ausgegeben.

```
void ausgeben( knoten *anfang );
```
 - Das Element, auf das `loeschElement` zeigt, soll gelöscht werden.

```
void loeschen( knoten *loeschElement,
              knoten **anfang, knoten **ende );
```
 - Das erste Element mit info-Wert `suchWert` soll gesucht werden. Wenn kein solches Element in der Liste ist, soll das Ergebnis NULL sein.

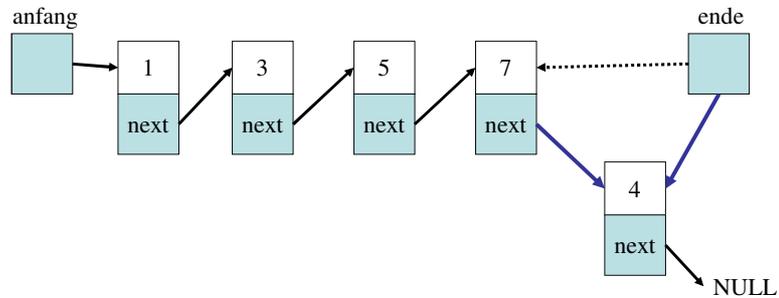
```
knoten * suchen( int suchElement, knoten *anfang );
```
 - Die Struktur wird global definiert.

Dirk Seeber , Informatik , Teil 5

36

Verkettete Listen - Einfügen

- Neues Element am Ende einfügen:



Dirk Seeber , Informatik , Teil 5

37

Verkettete Listen - Einfügen

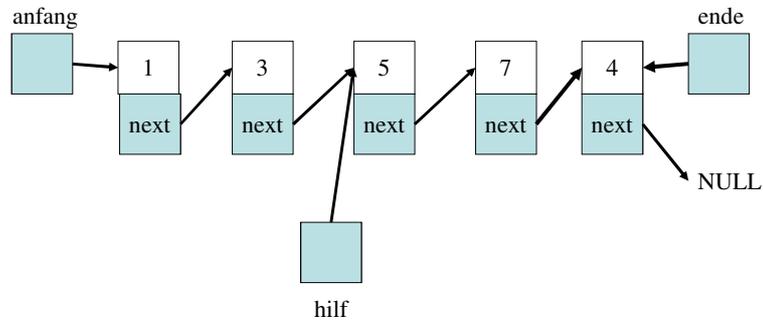
```
void einfuegen ( int x, ... )      // Einfuegen am Ende
{                                  // der Liste
    struct knoten *p = new knoten; // erzeuge neuen Knoten
    p->info = x;                   // Infowert wird x
    p->next = NULL;                // next-Wert wird mit
                                   // NULL initialisiert
    if ( NULL == *anfang )        // leere Liste
    {
        *anfang = p;
        *ende = p;
    }
    else                            // Liste nicht leer
    {
        (*ende)->next = p;        // Einfuegen am Ende
        *ende = p;                // neues Ende
    }
}
```

Dirk Seeber , Informatik , Teil 5

38

Verkettete Listen - Ausgeben

- Liste von Anfang bis Ende ausgeben:



Dirk Seeber , Informatik , Teil 5

39

Verkettete Listen - Ausgeben

```
void ausgeben ( ... )           // Ausgeben von Anfang
{                               // bis Ende
    struct knoten *p = anfang; // p zeigt auf aktuellen
                               // Knoten

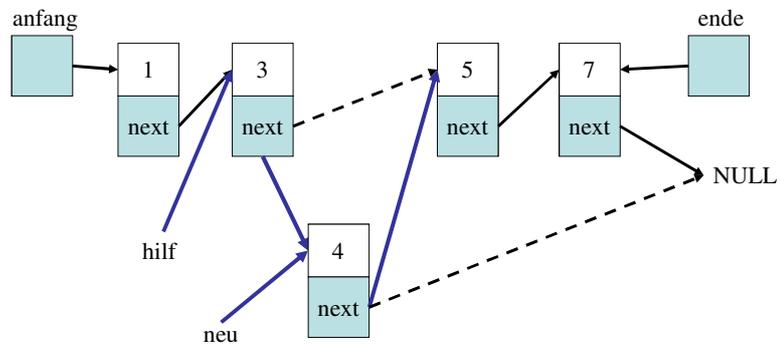
    cout << "Liste: ";
    while ( NULL != p)
    {
        cout << p->info << " "; // Infoteil des aktuellen
                                // Knoten ausgeben
        p = p->next;           // p auf Folgeelement setzen
    }
    cout << endl;
}
```

Dirk Seeber , Informatik , Teil 5

40

Verkettete Listen – Sortiert Einfügen

- Neues Element sortiert einfügen:



Dirk Seeber , Informatik , Teil 5

41

Verkettete Listen – Sortiert Eingeben

```
void sortiert_eingeben ( ... )
{
    struct knoten *p = anfang; // p zeigt auf Anfangsknoten

    // Fallunterscheidung (geschachtelt if-else)
    // 1. Einfügen am Anfang
    // 2. Einfügen am Ende
    // 3. Einfügen "Mittendrin"

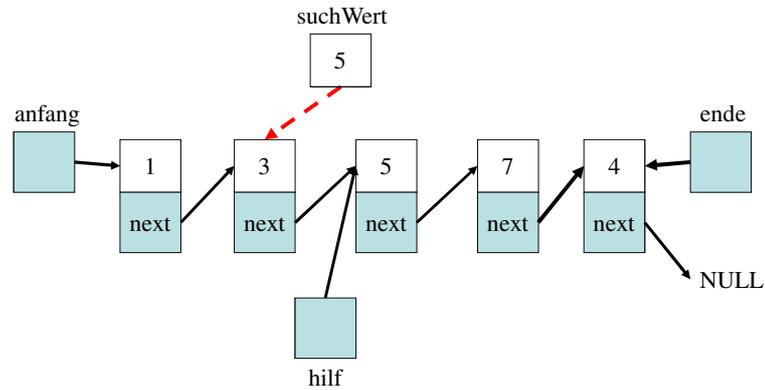
}
```

Dirk Seeber , Informatik , Teil 5

42

Verkettete Listen - Suchen

- Element in der Liste suchen:



Dirk Seeber , Informatik , Teil 5

43

Verkettete Listen - Suchen

```
struct knoten *suchen (int x, ...) // sequentielles suchen
{
    struct knoten *p = anfang; // p zeigt auf Anfangsknoten
    bool gefunden = false; // Startwert nicht gefunden!

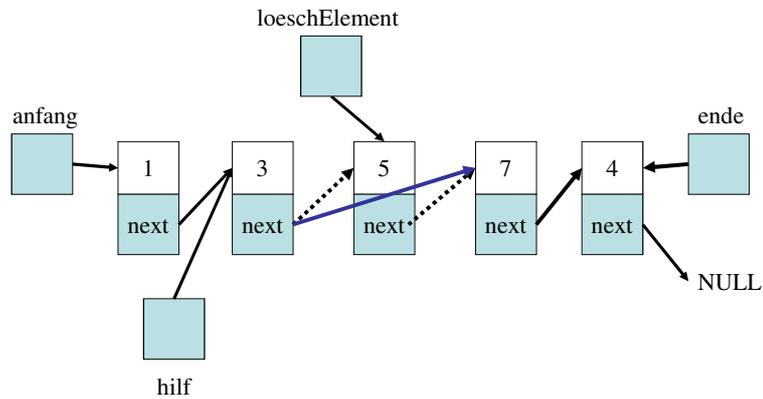
    while (( NULL != p) && (false == gefunden))
    {
        if ( p->info == x )
        {
            gefunden = true; // Wert gefunden
        }
        else
        {
            p = p->next; // p auf Folgeelement setzen
        }
    }
    return p; // Knoten ausgeben
}
```

Dirk Seeber , Informatik , Teil 5

44

Verkettete Listen - Löschen

- Element aus der Liste entfernen:



Dirk Seeber, Informatik, Teil 5

45

Verkettete Listen - Löschen

```
void loeschen ( knoten *p, ... )
{
    knoten *p1 = *anfang;      // p1 zeigt auf Anfangsknoten

    if ( NULL == *anfang )
    {
        cout << "leere Liste" << endl;    // leere Liste
    }
    else
    {
        while ( NULL != p1->next )
        {
            . . .
        }
    }

    // ACHTUNG!! Speicherfreigabe beachten!
}
```

Dirk Seeber, Informatik, Teil 5

46

Verkettete Listen - Hauptprogramm

```
int main()
{
    knoten *anfang = NULL;           // Leere Liste
    knoten *ende = NULL;           // Leere Liste
    int eingabe = 1;
    knoten *p;
    while (eingabe > 0)
    {
        cout << "Integer (Abbruch mit negativem Wert): ";
        cin >> eingabe;
        einfuegen(eingabe, ... );
    }
    ausgeben( ... );
    cout << "Integer, der gelöscht werden soll: ";
    cin >> eingabe;
    p = suchen(eingabe, ... );
    if ( NULL != p ) {
        loeschen(p, ... );
    }
    else {
        cout << eingabe << "nicht in Liste vorhanden" << endl;
    }
    ausgeben(...);
    return 0;
}
```

Dirk Seeber , Informatik , Teil 5

47

Typedef

- In C gibt es kein Datentyp `string` oder `bool`.
- Diese Datentypen kann der Programmierer selbst definieren. Es existiert die Möglichkeit, Namen zu definieren. Mit `typedef bestehenderTyp neuerName ;` kann dem bestehenden Typ ein neuer Name gegeben werden.
- Es wird - eigentlich - **kein** neuer Typ definiert, sondern nur ein neuer Name eingeführt.

Dirk Seeber , Informatik , Teil 5

48

Typedef

- **Beispiele:**

```
#include <iostream>
using namespace std;

typedef int BOOL;
#define TRUE 1
#define FALSE 0

int main()
{
    BOOL bool_var = TRUE;
    cout << "bool_var = " << bool_var << endl;

    bool_var = FALSE;
    cout << "bool_var = " << bool_var << endl;

    return 0;
}
```

Dirk Seeber , Informatik , Teil 5

49

Typedef

- **Beispiele:**

```
#include <iostream>
using namespace std;

typedef double real;

int main()
{
    real x = 1.2;
    cout << "x = " << x << endl;
    return 0;
}

oder
typedef enum wie {nicht, aufsteigend, absteigend};
```

Dirk Seeber , Informatik , Teil 5

50