

# Programmiersprache 1 (C++)

Prof. Dr. Stefan Enderle

NTA Isny

# 10. Dynamische Speicherverwaltung

# Scope / Sichtbarkeit

- Wird eine Variable in einer Funktion deklariert, so ist sie nur innerhalb dieser Funktion „sichtbar“.

Beispiel:

```
int add(int a, int b)
{
    int summe;           // Hilfsvariable
    summe = a + b;      // Summe berechnen
    return summe;       // Summe zurückliefern
}
```

- Der Speicherplatz für `summe` wird beim Eintritt in die Funktion angelegt und bei Verlassen wieder freigegeben.
- `summe` ist außerhalb der Funktion nicht „sichtbar“.

# Scope / Sichtbarkeit

- Problem / Frage:

Wie kann eine Funktion Speicherplatz anlegen, der nach Verlassen der Funktion NICHT gelöscht wird?

→ „Dynamische Speicherverwaltung“

# Wdh: Zeiger und Adressen

- Deklaration einer Variablen erzeugt Speicherplatz:

```
int variable; // liegt irgendwo im Speicher
```

- Die Adresse einer Variablen kann einem Zeiger zugewiesen werden:

```
int variable;  
int* zeiger = &variable; // Speicheradresse
```

- Die alleinige Deklaration eines Zeigers erzeugt noch keinen Speicherplatz!

```
int* zeiger; // Kein Speicherplatz für int !
```

# new

- Der Operator `new` reserviert Speicherplatz und liefert die Adresse davon zurück.
- Aufruf: `new <Typ>`
- Die Größe des benötigten Speicherplatzes entnimmt `new` dem übergebenen Typ.
- Dies funktioniert für elementare Datentypen (`int`, `float`, ...), für eigene Datentypen (`enum`, `struct`) und für Klassen (z.B. `string`).

# new - Beispiele

- Anlegen von Speicherplatz für einen `int`:

```
int* i;           // Zeiger deklarieren  
i = new int;     // Speicher anlegen
```

- Anlegen von Speicherplatz für einen `double`:

```
double* i;  
i = new double;
```

- Anlegen eines Objektes einer Klasse (`string`):

```
string* s;  
s = new string;
```

# new - Beispiele

- Anlegen einer eigenen Struktur:

```
struct Kunde                // Definition
{
    string name;
    string adresse;
};

Kunde* k = new Kunde;      // anlegen

k->name = "Hannes Maier";  // füllen
k->adresse = "Blumenweg 12, Isny";
```

# new - Initialisierung

- Elementare Datentypen sind nach Aufruf von new nicht initialisiert.
- Zur Initialisierung kann ein passender Anfangswert in Klammern angegeben werden:

```
i = new int(0);  
d = new double(17.5);  
s = new string("Hallo");
```

# new - Initialisierung

- Strukturen müssen elementweise initialisiert werden:

```
Kunde* k = new Kunde;    // anlegen
```

```
k->name    = "Hannes Maier";
```

```
k->adresse = "Blumenweg 12, Isny";
```

- `k->` entspricht `(*k)`.

# new [ ]

- Mit dem Operator `new [ ]` wird Speicherplatz für ein Array von Objekten reserviert.

- Anlegen von Speicherplatz für 100 `ints`:

```
int *i;           // Zeiger deklar.  
i = new int[100]; // Speicher anlegen
```

- Anlegen von 100 Strings:

```
string* s;  
s = new string[100];
```

# Zugriff auf dynamischen Speicher

- Auf Speicherplatz, der mit `new` reserviert wurde, wird über den entsprechenden Zeiger zugegriffen.
- Beispiel für elementaren Typ:

```
int *i;           // Zeiger deklar.  
i = new int;     // Speicher anlegen  
*i = 15;         // Wert zuweisen  
cout << *i << endl; // Wert ausgeben  
cout << i << endl;  // Adresse ausgeben!
```

# Zugriff auf dynamische Strukturen

- Zugriff auf Strukturen über den Pfeil-Operator:

```
Kunde* k = new Kunde;    // anlegen
```

```
k->name    = "Hannes Maier";
```

```
k->adresse = "Blumenweg 12, Isny";
```

```
cout << k->name << " "  
      << k->adresse << endl;
```

# Zugriff auf dynamische Arrays

- Beispiel für ein dynamisches Array:

```
int *i;           // Zeiger deklar.  
i = new int[100]; // Speicher anlegen  
i[3] = 15;       // Wert zuweisen  
cout << i[3] << endl; // Wert ausgeb.  
cout << i << endl; // Adresse ausgeben!
```

- **Bemerkung:** `s[n]` entspricht `*(s+n)`  
wobei für `n` die korrekte Größe des Objektes eingesetzt wird.

# delete

- Nach Benutzen des dynamischen Speichers muss dieser wieder freigegeben werden.
- Nicht freigegebener Speicher bleibt zum Ende des Programms reserviert und wird dann durch das BS wieder freigegeben.
- Der Operator `delete` gibt reservierten Speicherplatz wieder frei.
- Aufruf: `delete <Zeiger>`

# delete - Beispiele

- Anlegen und Freigeben eines `int`:

```
int* i;           // Zeiger deklarieren
i = new int;     // Speicher anlegen
...              // int benutzen
delete i;        // Speicher freigeben
```

- Anlegen und Freigeben eines `double`:

```
double* d;       // Zeiger deklarieren
d = new double; // Speicher anlegen
...              // double benutzen
delete d;        // Speicher freigeben
```

# delete [ ]

- Mit dem Operator `delete[]` wird Speicherplatz für ein Array von Objekten wieder freigegeben.
- Anlegen und Freigeben von 100 `ints`:

```
int *i;           // Zeiger deklar.  
i = new int[100]; // Speicher anlegen  
...             // ints benutzen  
delete[] i;      // Speicher freig.
```

# Scope / Sichtbarkeit

- Anfangsfrage:

Wie kann eine Funktion Speicherplatz anlegen, der nach Verlassen der Funktion NICHT gelöscht wird?

```
Kunde* kundeAnlegen(string name, string adresse)
{
    Kunde* k = new Kunde;

    k->name = name;
    k->adresse = adresse;

    return k;
}
```