

TDP004 - Objektorienterad programmering

Pekare, inre klasser och speciella
medlemsfunktioner

Pontus Haglund & Rasmus Jonsson

Institutionen för datavetenskap

- 1 Mål med föreläsningen
- 2 Pekare
- 3 Inre klasser
- 4 Speciella medlemsfunktioner
- 5 Slumpgenerering
- 6 Rekursion

- 1 Mål med föreläsningen
- 2 Pekare
- 3 Inre klasser
- 4 Speciella medlemsfunktioner
- 5 Slumpgenerering
- 6 Rekursion

Mål med föreläsningen

Efter föreläsningen skall studenten kunna:

- Förstå och arbeta med pekare.
- Visualisera datorns minne.
- Skapa och hantera inre klasser.
- Skapa och hantera speciella medlemsfunktioner.
- Slumpgenerering i C++.
- Lösa problem rekursivt i C++

- 1 Mål med föreläsningen
- 2 Pekare**
- 3 Inre klasser
- 4 Speciella medlemsfunktioner
- 5 Slumpgenerering
- 6 Rekursion

Pekare

Pekare



Pekare

Vad är en pekare?

```
char x{'A'};  
char y{'B'};
```

Pekare

Vad är en pekare?

```
char x{'A'};  
char y{'B'};
```

x 'A'

y 'B'

Pekare

Vad är en pekare?

```
char x{'A'};  
char y{'B'};  
char* ptr{}
```

x 'A'

y 'B'

Pekare

Vad är en pekare?

```
char x{'A'};  
char y{'B'};  
char* ptr{}
```

ptr 

x 

y 

Pekare

Vad är en pekare?

```
char x{'A'};  
char y{'B'};  
char* ptr{&x};
```

ptr 

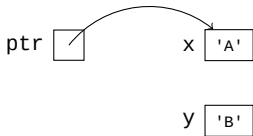
x 

y 

Pekare

Vad är en pekare?

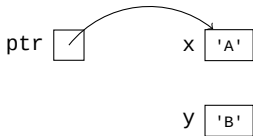
```
char x{'A'};  
char y{'B'};  
char* ptr{&x};
```



Pekare

Vad är en pekare?

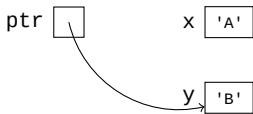
```
char x{'A'};  
char y{'B'};  
char* ptr{&x};  
ptr = &y;
```



Pekare

Vad är en pekare?

```
char x{'A'};  
char y{'B'};  
char* ptr{&x};  
ptr = &y;
```



Pekare

C++ pekare

```
int main()
{
    string text{"Hi"};
    string* pointer{&text};

    cout << "Address: "
         << pointer << "\n";
    cout << "Data: "
         << *pointer << endl;
}
```

Pointers

Hur använder jag en pekare?

```
string text{"Hi"};
string* pointer{&text};

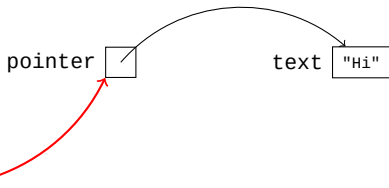
cout << "Address: "
      << pointer << "\n";
cout << "Data: "
      << *pointer << endl;
```



Pointers

Hur använder jag en pekare?

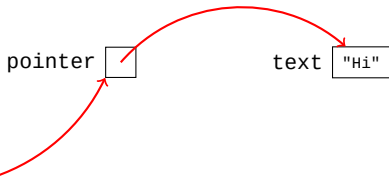
```
string text{"Hi"};  
string* pointer{&text};  
  
cout << "Address: "  
      << pointer << "\n";  
cout << "Data: "  
      << *pointer << endl;
```



Pointers

Hur använder jag en pekare?

```
string text{"Hi"};  
string* pointer{&text};  
  
cout << "Address: "  
      << pointer << "\n";  
cout << "Data: "  
      << *pointer << endl;
```



Pekare


Manuellt minne (heapen)

```
char* pointer{};
```

Pekare

Manuellt minne (heapen)


```
char* pointer{};
```

pointer 

Pekare

Manuellt minne (heapen)

```
char* pointer{new char{'A'}};
```

pointer 

'A' 

Pekare

Manuellt minne (heapen)

```
char* pointer{new char{'A'}};
```



Pekare

Manuellt minne (heapen)

```
char* pointer{new char{'A'}};
```



Pekare

Manuellt minne

```
{  
    char* pointer{new char{'A'}};  
}  
cout << "slut";
```


Pekare

Manuellt minne

```
{  
    char* pointer{new char{'A'}};  
}  
cout << "slut";
```

Pekare

Manuellt minne

```
{  
char* pointer{new char{'A'}};  
}  
cout << "slut";
```



Pekare

Manuellt minne

```
{  
  char* pointer{new char{'A'}};  
}  
cout << "slut";
```

'A'

Pekare

Manuellt minne

```
{  
    char* pointer{new char{'A'}};  
}  
cout << "slut";
```

'A'

Pekare

Manuellt minne - delete

```
{  
    char* pointer{new char{'A'}};  
    delete pointer;  
    pointer = nullptr;  
}  
cout << "slut";
```

Pekare

Manuellt minne - delete

```
{  
    char* pointer{new char{'A'}};  
    delete pointer;  
    pointer = nullptr;  
}  
cout << "slut";
```

Pekare

Manuellt minne - delete


```
{  
char* pointer{new char{'A'}};  
delete pointer;  
pointer = nullptr;  
}  
cout << "slut";
```



Pekare

Manuellt minne - delete


```
{  
  char* pointer{new char{'A'}};  
  delete pointer;  
  pointer = nullptr;  
}  
cout << "slut";
```

pointer 

Pekare

Manuellt minne - delete

```
{  
    char* pointer{new char{'A'}};  
    delete pointer;  
    pointer = nullptr;  
}  
cout << "slut";
```

pointer 

Pekare

Manuellt minne - delete

```
{  
    char* pointer{new char{'A'}};  
    delete pointer;  
    pointer = nullptr;  
}  
cout << "slut";
```

Pekare

Manuellt minne - delete

```
{  
    char* pointer{new char{'A'}};  
    delete pointer;  
    pointer = nullptr;  
}  
cout << "slut";
```

Pekare

Pekare och medlemsfunktioner

```
int main()
{
    string* pointer{ new string{"A string"} };

    cout << (*pointer).substr(0,3) << "\n";
    cout << pointer -> substr(0,3) << endl;

    delete new_pointer;
}
```

- 1 Mål med föreläsningen
- 2 Pekare
- 3 Inre klasser**
- 4 Speciella medlemsfunktioner
- 5 Slumpgenerering
- 6 Rekursion

Inre klasser

Inre klasser



Inre klasser

C++ inre klass

```
class Snake
{
private:
    struct Segment;
public:
    Snake(int x, int y);

    void print() const;
    void add(int x, int y);

private:
    Segment* head;
    struct Segment
    {
        int x;
        int y;
        Segment* next;
    };
};
```

Inre klasser

C++ inre klass

- Inre klasser kan vara smidigt om man behöver en klass som stöd för att lösa det övergripande problemet men inte vill att användaren ska skapa objekt av denna typ
- I Snake exempelvis vill vi bara att användaren skapar objekt av typen Snake men vi behöver objekt av typen Segment för att lösa vår länkade struktur.
- Inre klasser kan ofta vara aggregat vilket är fallet i Segment - Detta innebär att objekt av denna typ bara har publika datamedlemmar. De kan också ha Destruktor och Konstruktörer om detta behövs.

- 1 Mål med föreläsningen
- 2 Pekare
- 3 Inre klasser
- 4 Speciella medlemsfunktioner**
- 5 Slumpgenerering
- 6 Rekursion

Speciella medlemsfunktioner

Lista

```
int main()
{
    List my_list{};
    my_list.insert(1);
    my_list.insert(2);
}
```

Speciella medlemsfunktioner

Kopiering

```
int main()
{
    List my_list{};
    my_list.insert(1);
    my_list.insert(2);

    List copy{my_list};

    List new_list{};
    new_list.insert(3);

    new_list = copy;
}
```

Kopiering - header

Följande konstruktor och operator behöver deklarerars och definieras:

```
class List
{
    public:
        //...
        List(List const& other);

        List& operator=(List const& other);

    private:
        //...
}
```

Speciella medlemsfunktioner

Flytt

```
int main()
{
    List my_list{};
    my_list.insert(1);
    my_list.insert(2);

    List copy{ move(my_list) };

    List new_list{};
    new_list.insert(3);

    new_list = move(copy);
}
```

Flytt - header

Följande konstruktor och operator behöver deklarerars och definieras:

```
class List
{
    public:
        //...
        List(List && other);

        List& operator=(List && other);

    private:
        //...
}
```

Destruktor

Eftersom vi har datamedlemmar som är pekare behöver vi också deklarerera och definiera destruktorn:

```
class List
{
    public:
        //...
        ~List();

    private:
        //...
}
```

- 1 Mål med föreläsningen
- 2 Pekare
- 3 Inre klasser
- 4 Speciella medlemsfunktioner
- 5 Slumpgenerering**
- 6 Rekursion

Slumpgenerering

Random

```
int main()
// Exempel anpassat från en.cppreference.com
{
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> dis(1, 6);

    for (int n=0; n<10; ++n)
    {
        // Distributionen använder generatoren
        // för att välja ett tal i sitt intervall
        cout << dis(gen) << ' ';
    }
    cout << endl;
}
```

- 1 Mål med föreläsningen
- 2 Pekare
- 3 Inre klasser
- 4 Speciella medlemsfunktioner
- 5 Slumpgenerering
- 6 **Rekursion**

Rekursion

Kopiera ett träd

```
int main()
{
    Tree my_tree{1,2,3,4,5,6};

    Tree copy{my_tree};
}
```

www.liu.se