Pekare

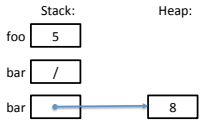
och minneshantering

Pekare

- Variabel
  - Namn
  - Värde
  - Datatyp
- Pekare == Variabel som lagrar en adress
  - Namn
  - Värde == En address
  - Datatyp == Adress till en viss datatyp
- Rita. Förklara! Ingen magi!

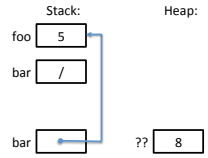
Pekarexempel

```
int foo = 5;
int* bar = nullptr;
bar = new int(8);
```



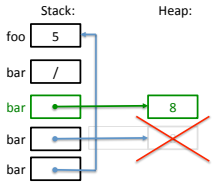
Pekarexempel

```
int foo = 5;
int* bar = nullptr;
bar = new int(8);
bar = &foo;
```



Pekarexempel

```
int foo = 5;
int* bar = nullptr;
bar = new int(8);
delete bar;
bar = &foo;
```



Pekaroperatorer

```
*  // content of some address
&  // address of some content
-> // member of (same as (*). )
[] // index of (same as *(+) )
+  // lite magi!

nullptr // C++11 nollpekare
0       // C++ nollpekare
NULL    // C nollpekare
```

Pekare och const

```
int* ptr; // inget const
const int* ptr; // adr till const int
int const* ptr; // adr till const int
int* const ptr; // const adr till int
int const* const ptr; // allt const
const int* const ptr; // allt const
const int const* ptr; // FEL
const int const* const ptr; // FEL
```

Pekarmakron

```
// Some macros if you prefer good names
#define address_of(var) (&var)
#define content_of(ptr) (*ptr)
#define member_of(ptr, mbr) (ptr->mbr)
#define member_of(ptr, mbr) ((*ptr).mbr)
```

Pekartemplates

```
template<typename V>
V* address_of(V& v)
{
  return &v;
}

template<typename P>
P& content_of(P* p)
{
  return *p;
}
```

Dynamiskt minne

- Allokering (”lån”)
  - new DATATYP(INITIERING);
  - new DATATYP;
    - skapar en ”anonym” variabel (utan namn)
    - lånar minne till variabeln
    - returnerar adressen där variabeln återfinns
    - konstruktor anropas!
- Avallokering (”återlämning”)
  - delete ADRESS;
    - återlämnar minnet för variabeln på adressen
    - adress eller minnesinnehåll ändras INTE!
    - minnet MÅSTE vara ”lånat” med new
    - allt lånat minne MÅSTE återlämnas EXAKT en gång
    - destruktor anropas!

Övning

Skriv om ett program (nästa slide) så att bara pekarvariabler nyttjas.

Visa skillnad på stack och heap.

Rita! Förklara!

Ett litet övningsprogram

```
int sum_and_swap(int* a, int* b)
{
  int save = *a;
  *a = *b;
  *b = save;
  return *a + *b;
}
int main()
{
  int a = 3;
  int b = 4;
  cout << sum_and_swap(&a, &b) << endl;
  cout << sum_and_swap(&a, &b) << endl;
  return 0;
}
```

Kontrollera minnesläckor

- Ubuntu:

```
sudo apt-get install valgrind
g++ -g my_program.cc
valgrind –tool=memcheck a.out
```

- Solaris:

```
g++ -g my_program.cc
bcheck a.out
```

- Dugga
- Laborationer
- mm adm

Minneshantering

```
#ifndef _POINTER_H_
#define _POINTER_H_

// Exempel på hur klasser stöder pekarhantering och djup kopiering
class Pointer
{
public:
  Pointer();  // Konstruktor (default)
  ~Pointer(); // Destruktor

  Pointer(Pointer const& p); // Kopieringskonstruktor
  Pointer(Pointer&& p);      // Flyttkonstruktor

  Pointer& operator=(Pointer const& rhs); // Kopieringstilldelning
  Pointer& operator=(Pointer&& rhs);      // Flyttoperator (flytt-tilldelning)

  int& operator*(); // Avreferering (”content of”)
private:
  int* ptr;
};

#endif
```

Minneshantering

```
// "Typiska" implementationer av minneshanterings-
// operationer i klasser som innehåller pekare

#include <algorithm> // för std::swap

Pointer()                 : ptr(new int)     {}
~Pointer()                                   { delete ptr; }

Pointer(Pointer const& p) : ptr(new int(*p.ptr)) {}
Pointer(Pointer&& p)      : ptr(nullptr)     { std::swap(ptr, p.ptr); } //C++11

Pointer& operator=(Pointer const& rhs)
{
  if (this != &rhs) // undvik onödigt jobb vid självtilldelning
  {
    Pointer copy(rhs);       // nyttja kopieringskonstruktor
    std::swap(this->ptr, copy.ptr); // nyttja destruktor för 'copy'
  }
  return *this; // this är en pekare till klassvariabeln funktionen anropades "på"
}
Pointer& operator=(Pointer&& rhs)           { std::swap(this->ptr, rhs.ptr); } //C++11

int& operator*()                            { return *ptr; }
```

Enkel länkad lista

”Enkel länkad lista” blev en särskrivning

Apropå: Ibland blir det fel med särskrivning:

- En kort vuxen burn hårig sjuk sköterska tvättar barn under kläder
- kassa personal sökes
- kassa skåp säljes

Enkellänkad lista

- Rita! Förklara!

- class List
- class List::Link
- sätt in 5, 3, 9, 7

- Försök utöka listan utan att använda dynamiskt minne – varför går det inte?

Lista i verkligheten

- Varje länk är en person
- Höger hand håller värdet
- Vänster hand håller fast nästa länk

- Släpps en länk ”seglar” den iväg som en heliumballong...

Inbyggda arrayer

```
// C-array eller inbyggd array:
int array[5] = {1, 2, 3, 4, 5};
int* arrptr = array;

arrptr = new int[10]; // OBS! []
delete[] arrptr;      // OBS! []

// Rita! Förklara!
// Storlek?
// Lagras inte någonstans (om inte du gör det!)
```

C-sträng, C-array

const char* cstring = ”Kalle”; // vanligen
char const* cstring = ”Kalle”; // jag?

// egentligen:
const char data[] = {’K’, ’a’, ’l’, ’l’, ’e’, ’\0’};
const char* cstring = data;

Kommandoradsargument

```
int main(int argc, char* argv[])
{
  vector<string> arg(argv, argv+argc);

  for (unsigned i = 0; i < arg.size(); ++i)
  {
    cout << arg[i] << endl;
  }
  return 0;
}
```