

# Makefiles

---

- C-projekt blir snabbt stora
  - Många källkodsfiler
  - Bibliotek spridda på olika ställen
  - Olika konfigurationer (t.ex. via `#defines`)
- Att manuellt skriva kommandona för att bygga systemet blir invecklat, pga. många långa parametrar, ex:

```
gcc -I/usr/local/include/ -  
L/usr/local/lib -DDEBUG -DWIN32  
myfile.c -g -O2 -Wall -o myfile.o -lm  
-lcrypt -ltext
```

# Makefiles

---

- Istället bygger man en s.k. Makefile som innehåller regler för hur ett programpaket skall byggas
- Baserar sig på
  - definitioner, tex.
    - kataloger där filer finns
    - flaggor / parametrar
  - beroendeförhållanden, tex.
    - att en viss objektfil byggs från vissa källkodsfiler
    - vissa system byggs utgående från vissa objektfiler
  - regler för hur man bygger komponenter
    - hur en kompilator / linker skall aktiveras

# Makefile - exempel

---

```
#-----Macros-----  
  
# for cs machines  
#BASEDIR = /usr/project/courses/cps008/lib  
# for acpub machines  
BASEDIR = /afs/acpub.duke.edu/users8/ola/courses/lib  
  
TLIB = $(BASEDIR)/libtapestry.a  
INCLUDES = -I. -I$(BASEDIR)  
  
# set up compiler and options  
CXX = g++  
CXXFLAGS = -g $(INCLUDES)
```

# Makefile - exempel

---

```
#-----Suffix Rules-----  
# set up C++ suffixes and relationship between  
  .cc and .o files  
  
.SUFFIXES: .cc  
  
.cc.o:  
  -> $(CXX) $(CXXFLAGS) -c $<  
  
.cc :  
  -> $(CXX) $(CXXFLAGS) $< -o $@ -lm $(TLIB) -  
  lg++  
  NOTE!! → correspond to a TAB
```

# Makefile - exempel

---

```
#-----File Dependencies-----  
SRC = application.cc menu.cc menuitem.cc pixmap.cc  
      usepix.cc readcommand.cc quitcommand.cc filelister.cc  
      templateapp.cc displaycommand.cc  
OBJ = $(addsuffix .o, $(basename $(SRC)))  
usepix: $(OBJ)  
    -> $(CXX) $(CXXFLAGS) -o $@ $(OBJ) -lm $(TLIB) -  
      lg++  
  
#-----Other stuff-----  
depend:  
    -> makedepend $(CXXFLAGS) -Y $(SRC)  
clean:  
    -> rm -f $(OBJ)
```

# Makefiles - delar

---

- Makrodefinitioner – text-substitueringar
  - BASEDIR = /export/home/appdev/
  - Används via \$BASEDIR
- Specialkommandon
  - OBJ = \$(addsuffix .o, \$(basename \$(SRC)))
  - Byter ut alla filsuffix mot .o

# Makefile - delar

---

- Beroendeförhållanden

```
usepix: $(OBJ)
```

```
-> $(CXX) $(CXXFLAGS) -o $@ $(OBJ) -lm $(TLIB) -lg++
```

- Obs ”->” svarar för tab-tecknet
- Dvs. usepix är beroend av alla objektfiler, och raden under berättar hur man skall bygga ”usepix”
- Från kommandoraden aktiverar man genom att skriva

```
$ make usepix
```

# Speciella beroenden

---

- Första beroendeförhållande aktiveras som "default" (dvs bara "make" på kommandoraden)
- Pseudo-targets: Egentlig målfil finns ej, men kan användas t.ex. för att bygga andra mål  
`testsuite: test1 test2`
- Mål utan beroenden kan användas för att exekvera vissa kommandon  
`clean:`  
`-> rm -f $(OBJ)`



---

# C++

# C++ Material

– Stephen Prata: C++ Primer Plus 5/e, Sams Publishing

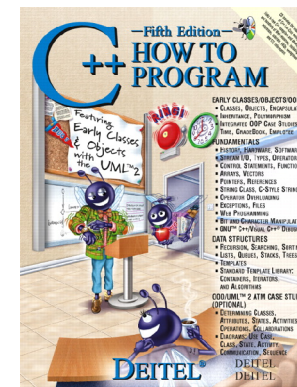


– Frank Brokken: C++ Annotations:  
<http://www.icce.rug.nl/documents/cplusplus/>

C++ Annotations Version 6.2.3

Frank B. Brokken  
Computing Center, University of Groningen  
Nettelbosje 1,  
P.O. Box 11044,  
9700 CA Groningen  
The Netherlands  
Published at the University of Groningen  
ISBN 90 367 0470 7

– (Harvey & Paul) Deitel & Associates: C++ How to Program, 5/E, ISBN: 0-13-185757-6, Prentice Hall, 1536pp



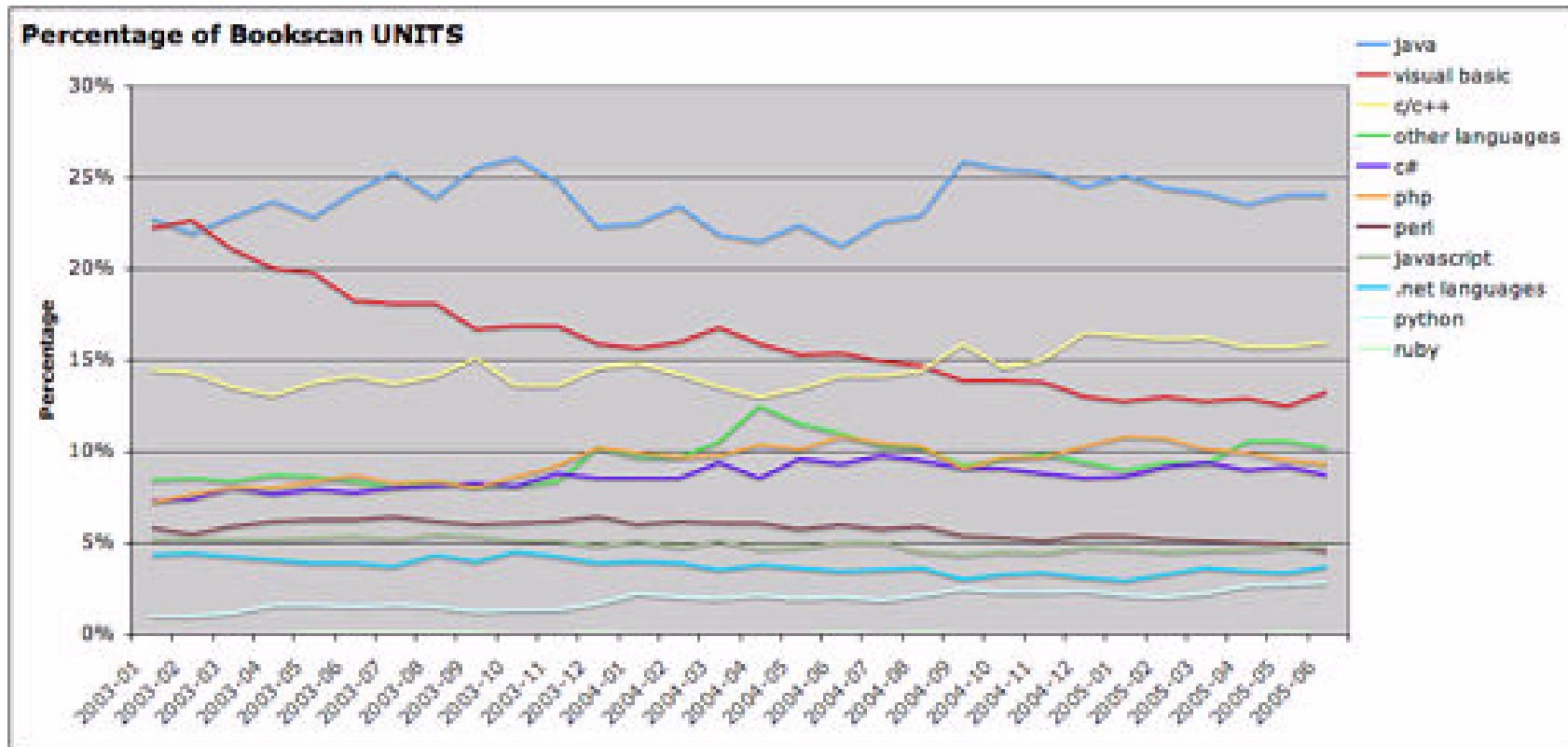
1994 - 2005

# Historia för C++

---

- 1980-talet: AT&T:s egen prekompilator, skriven av Bjarne Stroustrup
  - Kompilerade C++-syntaxen till standard C
- Idag de-facto-standard i industrin, grovt taget
  - Operativsystem skrivs i C
  - Applikationsprogram och plattformar skrivs i C++

# Sålda programmeringsböcker



# Fördelar ??

---

- Kod är lättare att återanvända → snabbare utveckling av nya program
- Det är enklare att skapa och använda nya datatyper
- Minnesanvändningen är enklare och mer transparent
- Program är mindre fel-benägna, pga striktare typer och typkontroll
- Gömmande av data är lättare

# Objekt-orienterat

---

- C++ inför objekt-orienterad programmering till C
  - klasser och deras instanser: objekt
    - data
    - metoder, som arbetar på datan
  - ärvning, polymorfism

# Skillnad C $\leftrightarrow$ C++

---

- Ganska långt kan C++ ses som ett superset av C, några skillnader jämfört med C
  - strikt typkontroll – funktioner måste vara deklarerade
  - typecasts:
    - `int(expression)`
    - `static_cast<type>(expression)`
      - Vanlig typändring (vid kompilering)
    - `const_cast<type>(expression)`
      - För att radera egenskaperna `const` / `volatile`
    - `reinterpret_cast<type>(expression)`
      - Förändring mellan inkompatibla typer (t.ex. pekare)
    - `dynamic_cast<type>(expression)`
      - Typändring (körtida) mellan kompatibla pekartyper, så att pekaren är valid, fel här -> NULL-pekare
  - `#define __cplusplus` – symbolen `__cplusplus` är definierad
- C++ overhead märks först då C++ egenskaper tas i användning

# C-funktioner i C++

---

- Normala C-funktioner, som kompilerats och finns i bibliotek, kan användas i C++, bara de deklarerats som C-funktioner

```
extern "C" void *malloc(unsigned);
```

- Alternativt kan de deklareraras

```
extern "C" {  
    void *malloc(unsigned);  
}
```

- Headers för både C och C++

```
#ifndef __cplusplus  
extern "C" {  
#endif  
    void *malloc(unsigned);  
#ifndef __cplusplus  
}  
#endif  
§
```

- Ofta i praktiken

```
#ifndef __cplusplus  
# define __BEGIN_DECLS extern "C" {  
# define __END_DECLS }  
#else  
# define __BEGIN_DECLS  
# define __END_DECLS  
#endif
```



# HelloWord i C++

---

```
// HelloWorld.cpp - OBS fileextension ".cpp"  
#include <stdio.h>
```

```
int main() {  
    printf("HelloWorld!\n");  
    return 0;  
}
```

```
% g++ HelloWorld.cpp -o HelloWorld
```

```
% ./HelloWorld
```

```
HelloWorld!
```

*Ser ju ut som C.... så vad är idén???*

# Nytt i C++

---

- Lokala variabler -- deklaration
  - C: Endast i början av funktioner / funktionella block
  - C++: Var som helst i koden (men även ANSI C99)
- Namnrymder
  - Symboler definieras i en större context

```
using namespace std;
```
- Överladdade funktioner (function overloading)
  - Det kan finna flere funktioner med samma namn men med olika prototyper

```
void show(int val) {printf("Int: %i\n", val);}
void show(float val) {printf("Float: %f\n", val);}
void show(char *val) {printf("String: %s\n", val);}
```

# Nytt i C++ II

---

- Förvalda (default) funktionsargument

```
void showstring (char *val="Förvald");  
main() {  
    showstring("Explicit sträng");  
    showstring(); // Samma som showstring("Förvald");  
}
```

- Keyword "typedef"

- Behövs inte längre vid strukturer, unioner eller enum

```
struct mystruct {  
    int a;  
    float b;  
};  
mystruct my_var;
```

# Const i C++

---

- Nyckelordet "const"

- Även om "const" finns i C, används det ganska sällan. I C++ är det vanligt.

```
char * const str; (1)
```

```
char const * str; (2)
```

```
const char * str; (3)
```

- Regel: Läs från variabelnamnet mot vänster

- (1) Str är en konstant pekare till tecken
- (2) Str är en pekare till konstant tecken
- (3) Str är en pekare till tecken som är konstant

- Notera. (2) och (3) är ekvivalenta

```
*str = 'a';      /* (1): OK, (2)(3): FEL! */
```

```
str++;          /* (1): FEL!, (2)(3): OK */
```

# Referenser

---

- C++, som tillägg till variabler och pekare till variabler, "referenser" eller synonymer för variabler

```
int value;  
int &ref=value; /* Referens */  
value++;      /* Alt 1 */  
ref++;       /* Alt 2 */
```

- Referenser möjliggör att överföra funktionsargument som kan modifieras (dvs. C:s "call by value" gäller inte mera)

# Referenser II

---

- I C endast

```
void increase(int *valp) {  
    *valp += 5;  
}  
increase(&my_var); /* Måste ge address */
```

- I C++

```
void increase(int &val) {  
    val += 5;  
}  
increase(my_var); /* Argumentet blir  
    automatisk en referens !! */
```

# När använda referenser???

---

- När funktionen ej modifierar det som överförs, kan referens användas
- Annars rekommenderas att pekare används

```
struct Point {
    int x,y;
};
void dispPoint(Point const &p) {
    printf("(x,y)=(%i,%i)\n", p.x,p.y);
}
void movePoint(Point *p) {
    p->x +=5;
}
main() {
    Point myPoint = {3,7};
    movePoint(&myPoint);
    dispPoint(myPoint);
}
```

# Funktioner i strukturer

---

- Funktioner kan i C++ vara delar av strukturer, de kallas då *metoder*

```
struct Person {
    char m_name[50];
    char m_address[50];
    void print();
};
void Person::print() {
    printf("Name: %s, Address: %s\n", m_name, m_address);
}
main() {
    Person p;
    strcpy(p.m_name, "Axel Eklund");
    strcpy(p.m_address, "Axelborg");
    p.print();
}
```



# Nya datatyper

---

- `bool` – datatyp som endast tar värdena `true` och `false`
- `wchar_t` – datatyp för "wide" tecken (vanligen 16 bit)
  - Inmatning via: `L"mytext"`
- `string` – via biblioteket `<string>`  
stränghantering via klasser

```
string s="Testing string";
s += " Adding to string";
printf("%s\n", s.c_str());
```

# Namnrymder

---

- I C++ definierar man namnrymder
  - symboler kan definieras i namngivna namnrymder

```
namespace ListImp {  
    int AddToList(LIST *list, char *);  
};  
/* Användning */  
using ListImp::AddtoList;           /* Del av namnrymden */  
AddToList(l, "testing");  
ListImp::AddToList(l, "testing");  /* Ekvivalent */  
namespace NS = ListImp;  
NS::AddToList(l, "testing");       /* Ekvivalent */  
  
using namespace ListImp;           /* Hela namnrymden */
```

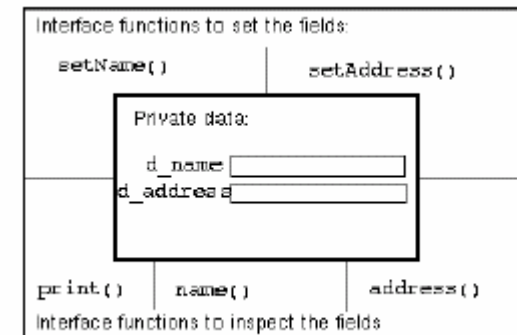
- ibland måste den globala namnrymden användas,  
"::" är namnresolutionsoperatören

```
float b=::cos(0.3412);
```

# Data hiding

---

- C++ erbjuder möjligheter att "gömma" data, det sker via nyckelorden `private`, `protected` och `public`
  - `public`: Fält som följer kan accesseras av all kod
  - `private`: Fält som följer kan accesseras endast av kod som hör till samma struktur
  - `protected`: Fält som följer kan accesseras av samma struktur, samt alla som härletts från denna



# Exempel

---

```
struct Person {
private:
    char m_name[50];
    char m_address[50];
public:
    void print();
    void setName(char const *p);
    void setAddress(char const *p);
};

void Person::print() {
    printf("Name: %s, Address: %s\n", m_name, m_address);
}

void Person::setName(char const *p) {
    strcpy(m_name, p);
}

main() {
    Person p;
    /* strcpy(p.m_name, "Axel Eklund"); */ //Illegal
    p.setName("Axel Eklund");
    p.print();
}
```

# Jfr med C-strukturer

---

- C:

```
typedef struct {
    char m_name[50];
    char m_address[50];
} PERSON;
/* Function som manipulerar,
   OBS måste ge pekare till
   struktur */
void print(PERSON const *p);
main() {
    PERSON p;
    print(&p);
}
```

- C++:

```
struct Person {
    char m_name[50];
    char m_address[50];
    print();
};
/* Function som manipulerar,
   pekare "implicit" */
void Person::print();

main() {
    Person p;
    p.print();
}
```

# En C++ klass

---

```
class CVehicle {
public:                                /* Synlighet: alla */
    void SetWeight(float w);          /* Metod */
private:                               /* Synlighet: klassmedlemmar */
    float m_weight;                  /* Data */
};

/* Implementation */
void Cvehicle::SetWeight(float w) {
    m_weight = w;
}

main() {
    Cvehicle veh;                      /* Instantiering */
    veh.SetWeight(4.33);
}
```

- **Jämfört med C: En klass är en struktur som innehåller både data och funktioner (=metoder) (i C endast data)**

# Så, vad är skillnaden mellan en struktur och en klass??

---

- I en struktur är den förvalda synligheten

”public”.

```
struct Person {  
    char m_name[50];  
    char m_address[50];  
    print();  
};
```

```
struct Person {  
public:  
    char m_name[50];  
    char m_address[50];  
    print();  
};
```

- I en klass är den förvalda synligheten

”private”

```
class Person {  
    char m_name[50];  
    char m_address[50];  
    print();  
};
```

```
class Person {  
private:  
    char m_name[50];  
    char m_address[50];  
    print();  
};
```

This document was created with Win2PDF available at <http://www.win2pdf.com>.  
The unregistered version of Win2PDF is for evaluation or non-commercial use only.  
This page will not be added after purchasing Win2PDF.