CS480

Ch 7

Handling Data

March 18, 2013

Chapter 7

Do as much as we can, come back to it later

- Handling data at runtime
  - static: pages 395-429
  - dynamic: pages 440-446
  - how does garbage collection work?
  - what is reference counting?
  - dynamic vs static binding
- Handling data at compile time (Symbol Table)
  - pages 429-440

Process Layout

Binding

- Static Binding

- Dynamic Binding

- Dynamic Dispatch

- RTTI: Run Time Type Identification (C++)

How does this work?

```
void simple() { … };


int main()
{
  ExampleClass *pExample = new ExampleClass();
  simple();
  pExample->foo();
  pExample->bar();
}
```

```
class ExampleClass
{
  public:
    void foo();
    virtual void bar();
};
```

```
class animal {
  private:
    int weight;
  public:
    animal() ;
    animal(int w);

    virtual void makeSound() = 0;
    virtual int getWeight2() = 0;
    virtual void hi();
    int getWeight() ;
    virtual int eat(animal &pA);
    void setWeight(int w);
    void setWeight(int *w);
    void setWeight(int &w);
};
class cat : public animal
{
  private:
    int weight;
  public:
    cat(int w);

    void makeSound();
    int getWeight2();
    void bye();
    void boggle();
    void boggle(int x);
};
```

If code is defined in the .h files the following output changes.

```
int main()
{
  animal *pAn = new cat(93);
  pAn->makeSound();

  printf("\ngetWeight: %d\n",
    pAn->getWeight());

  printf("getWeight2: %d\n",
    pAn->getWeight2());

  pAn->hi();
  //pAn->bye();     // ????
}
```

Code Available as Class Heirarchy Eclipse Project on schedule.

g++ -fdump-class-hierarchy-all

Display only the virtual table
http://www.codesourcery.com/public/cxx-abi/abi.html#vtable

```
Vtable for animal
animal::_ZTV6animal: 6u entries
0     (int (*)(...))0
8     (int (*)(...))(& _ZTI6animal)
16    __cxa_pure_virtual
24    __cxa_pure_virtual
32    animal::hi
40    animal::eat
```

```
Vtable for cat
cat::_ZTV3cat: 6u entries
0     (int (*)(...))0
8     (int (*)(...))(& _ZTI3cat)
16    cat::makeSound
24    cat::getWeight2
32    cat::hi
40    animal::eat
```

nm --demangle bin/animal.o

```
0000000000000042 T animal::hi()
00000000000000c2 T animal::eat(animal&)
0000000000000060 T animal::getWeight()
000000000000008a T animal::setWeight(int*)
00000000000000a6 T animal::setWeight(int&)
0000000000000072 T animal::setWeight(int)
0000000000000020 T animal::animal(int)
0000000000000000 T animal::animal()
0000000000000020 T animal::animal(int)
0000000000000000 T animal::animal()
0000000000000000 V typeinfo for animal
0000000000000000 V typeinfo name for animal
0000000000000000 V vtable for animal
                 U vtable for __cxxabiv1::__class_type_info
                 U __cxa_pure_virtual
                 U printf
```

nm bin/animal.o

http://www.codesourcery.com/public/cxx-abi/abi.html#mangling

```
0000000000000042 T _ZN6animal2hiEv                    void
00000000000000c2 T _ZN6animal3eatERS_                 reference S_
0000000000000060 T _ZN6animal9getWeightEv
000000000000008a T _ZN6animal9setWeightEPi            pointer to int
00000000000000a6 T _ZN6animal9setWeightERi            reference to int
0000000000000072 T _ZN6animal9setWeightEi             int
0000000000000020 T _ZN6animalC1Ei                     C1 complete obj ctor
0000000000000000 T _ZN6animalC1Ev
0000000000000020 T _ZN6animalC2Ei                     C2 base obj ctor
0000000000000000 T _ZN6animalC2Ev
0000000000000000 V _ZTI6animal
0000000000000000 V _ZTS6animal
0000000000000000 V _ZTV6animal
                 U _ZTVN10__cxxabiv117__class_type_infoE
                 U __cxa_pure_virtual
                 U printf
_ZN6animal9setWeightEi
_Z all mangled names start with _Z          N nested name
6animal (LengthName) 9setWidth (LengthName) E end marker i int param
```

GCC & Multiple Construtors

The constructor with "C1" in the linkage name is the complete object constructor. Your program calls this constructor when it creates an object whose complete type is A, such as "new A".

The constructor with "C2" in the linkage name is the base object constructor. Your program calls this constructor when it creates an object derived from A, such as "new B". Your program does *not* call the base object constructor for "new A".

`http://www.cygwin.com/ml/gdb/2004-07/msg00163.html`

nm --demangle bin/cat.o

```
000000000000006c T cat::getWeight2()
000000000000009c T cat::hi()
000000000000007e T cat::bye()
00000000000000d8 T cat::boggle(int)
00000000000000ba T cat::boggle()
0000000000000032 T cat::makeSound()
0000000000000000 T cat::cat(int)
0000000000000000 T cat::cat(int)
                 U animal::eat(animal&)
                 U animal::animal()
0000000000000000 V typeinfo for cat
                 U typeinfo for animal
0000000000000000 V typeinfo name for cat
0000000000000000 V vtable for cat
                 U vtable for __cxxabiv1::__si_class_type_info
                 U printf
```

nm --demangle bin/main.o

```
                 U _Unwind_Resume
                 U cat::cat(int)
                 U animal::getWeight()
                 U operator delete(void*)
                 U operator new(unsigned long)
                 U __gxx_personality_v0
0000000000000000 T main
                 U printf
```

```
_Unwind_Resume -- private C++ error handling method
__gxx_personality_v0 used to help unwind the stack
http://www.codesourcery.com/public/cxx-abi/abi-eh.html#base-personality
```

nm bin/main.o

```
                 U _Unwind_Resume
                 U _ZN3catC1Ei
                 U _ZN6animal9getWeightEv
                 U _ZdlPv
                 U _Znwm
                 U __gxx_personality_v0
0000000000000000 T main
                 U printf
```

Why are main() and print not mangled?

http://www.codesourcery.com/public/cxx-abi/abi.html#mangling

Java

- In Java, every method that is not **private** or **final** is implicitly virtual

Animal.java

```java
package edu.pacificu.cs.chadd;

public abstract class Animal {

  private int  weight;

  public Animal()
  {
    weight = -1;
  }

  public Animal(int w)
  {
    weight = w;
  }

  abstract public void makeSound();

  abstract public int getWeight2();

  public void hi()
  {
    System.err.println("hi");
  }
```

Cat.java | Zoo.java

```java
package edu.pacificu.cs.chadd;

public class Cat extends Animal {

  private int weight;

  public Cat(int w)
  {
    weight = w;
  }

  public void makeSound()
  {
    if (weight > 20)
    {
      System.err.print("Roar!");
    }
    else
    {
      System.err.print("meow");
    }
  }
  public void hi()
  {
    System.err.print("hi ");
    makeSound();
    System.err.println("!");
  }
```

```java
Cat cat = new Cat(93);
Animal animal = cat;

animal.makeSound();
animal.hi();
```

Javadoc | Declaration | Console

```
<terminated> Zoo [Java Application] /usr/lib64/jvm
Roar!hi Roar!!
```

Posted on class schedule

Data Layout

- Alignment

- Padding

- Packing

Data Layout and Padding

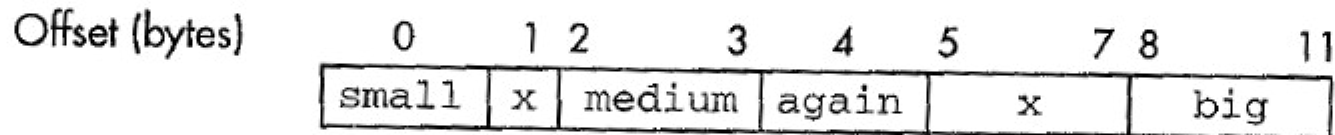

FIGURE 11.1 Structure layout and padding in memory

```
struct foo {
    char small;
    short medium;
    char again;
    int big;
}
```

Sweetman, <u>See MIPS Run</u>, page 310

Data Layout and Packing

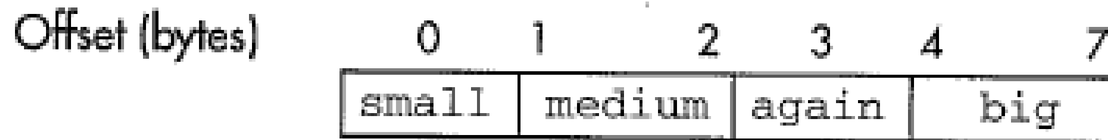

FIGURE 11.2 Data representation with #pragma pack(1)

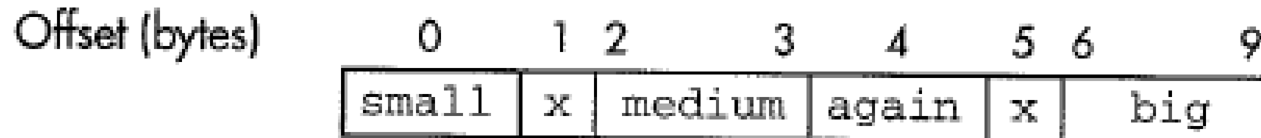

FIGURE 11.3 Data representation with #pragma pack(2)

Sweetman, <u>See MIPS Run</u>, page 312

Data Allocation

- Static

- Stack

- Heap
  - dangling references

C to malloc

- C language defines functions
  - malloc() / free() / calloc()
- A library (libc.so / glibc.so / msvcr100.dll) provides these functions
  - track the heap
  - reuse free() memory
- The kernel provides supporting functions
  - allocate more pages of memory (brk())

Linux Kernel: brk

- tracked by the Process Control Block in the Kernel (sched.h)
  - inside the mm_struct
- sys_brk() is a system call exported by the kernel to change brk.
  - mm/mmap.c

    ```
    SYSCALL_DEFINE1(brk, unsigned long, brk)
    ```
  - mm/nommu.c

Heap ftp.gnu.org/gnu/glibc/

C / gcc / Linux kernel / Linux loader / glibc

- _end: symbol that points to the first address after the data section (the start of the heap)
  - the *brk* points to the last address in the data segment (end of the heap)
  - **malloc** calls **brk()/sbrk()**
  - unsafe to mix **malloc()** and **brk()/sbrk()** on some systems

```
0000000000601018 A __bss_start
0000000000601008 D __data_start
0000000000601018 A _edata
0000000000601028 A _end
00000000004005e8 T _fini
00000000004003c0 T _init
0000000000400400 T _start
00000000004004e4 T main
```

```
#include <unistd.h>

int brk(void *addr);

void *sbrk(intptr_t increment);
```

Dynamic Memory

- Heap
  - malloc()

- Garbage Collection
  - reference counting
  - marking
  - dangling references

Parameter Passing

- Call by Value
- Call by Reference
- Copy Restore
- Call by Name (copy rule of Algol (1960))

```
int foo(int formal);

foo(actual);
```