C++ Report

MPW C++ and You

Tim Swihart

Development Tools
Product Marketing MPW
C++ Product Manager

Session Overview

- What, Why, Where,...
  - Tim Swihart, MPW C++ Product Mgr.
- New Features
  - Preston Gardner, C++ Lead Engr.
- MPW C++ and MacApp
  - Jack Palevich, Adv. Tech. Group
- Questions & Answers
  - Entire panel

What is MPW C++?

- Apple’s implementation of AT&T C++ Release 2.0
- Enhancements:
  - Fully supports Macintosh Toolbox
  - Complex Library redone using SANE
  - C++ source–level debugging w/SADE
  - Supports Object Pascal functions and procedures (á la MacApp)

Why C++ for MPW?

- Provides object technology for C programmers
  - Reduces development time
  - Increases application reliability
  - Facilitates reusable code
  - Makes application maintenance easier
  - Provides better model than procedural programming

C++ and MacApp

- Object Technology builds on a class library
- MacApp has years of use and testing behind it
- MPW C++ users can take full advantage of third-generation class library
- Today’s developers can leverage off of yesterday’s

Where Do I Get MPW C++?

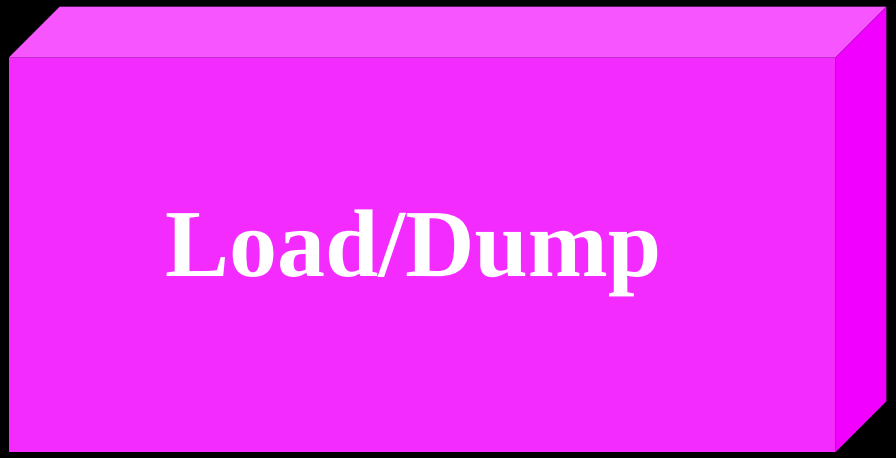
- Only from APDA
  - 1-800-282-2732 (U.S.)
  - 1-800-637-0029 (Canada)
  - 1-408-562-3910 (International)
- Price: $175
- Requires at least MPW 3.0 and MPW C 3.0*

* MPW 3.1 and MPW C 3.1 preferred

Preston Gardner

Development Systems Group
C++ Lead Engineer

MPW C++
New Features


Load/Dump

Why Load/Dump?

- C++ promotes code reuse through classes and type derivation
- But...
  - This forces even small programs to have big header files
  - The “hello world” program at the start of the C++ book uses about 2500 lines of header files

So...

- MPW C++ can now compile the headers and “dump” the compiler state to a file
- Unless the headers are changed, the compiler state can always be loaded from the dumped file
- Speed improvement: 2–3 X

```
// this file is "MonkeySink.cp"


#include "Monkey.h"
#include "Helicopter.h"
#include "KitchenSink.h"
main()
{ Monkey cheetah;
  Helicopter chopper;
  KitchenSink kelvinator;
  cheetah.learn_to_fly;
  chopper.hijack(cheetah);
  chopper.crash_land_in(kelvinator);}
```

```
// this file is "MonkeySink.cp"


#include "Monkey.h"
#include "Helicopter.h"
#include "KitchenSink.h"

main()
{ Monkey cheetah;
  Helicopter chopper;
  KitchenSink kelvinator;
  cheetah.learn_to_fly;
  chopper.hijack(cheetah);
  chopper.crash_land_in(kelvinator);}
```

```
// This file is "MonkeySinkDump.h"


#include "Monkey.h"
#include "Helicopter.h"
#include "KitchenSink.h"
```

// This file is "MonkeySinkLoad.cp"

```
// #include "Monkey.h"
// #include "Helicopter.h"
// #include "KitchenSink.h"
main()
{ Monkey cheetah;
  Helicopter chopper;
  KitchenSink kelvinator;
  cheetah.learn_to_fly;
  chopper.hijack(cheetah);
  chopper.crash_land_in(kelvinator);}
```

```
# Original command line:
CPlus -I HD:Hdrs MonkeySink.cp


# Command line to make dump file:
CPlus -I HD:Hdrs -dump MHS.dump ∂
MonkeySinkDump.cp


# Command line to load and make program:
CPlus -load MHS.dump MonkeySinkLoad.cp
```

```
/* This file is monkey.h */


#ifndef __MONKEY__
#define __MONKEY__ 1

class Monkey {
   .
   .
   .
};
#endif __MONKEY__
```

```
 /* This is MonkeySink.cp */

#ifndef   __MONKEY__
#include "Monkey.h"
#endif    __MONKEY__

#ifndef   __HELICOPTER__
#include "Helicopter.h"
#endif    __HELICOPTER__
#ifdef    __KITCHENSINK__
#include "KitchenSink.h"
#endif    __KITCHENSINK__
main()
{// ...etc...}
```

Things to Look Out for

- Code in header files–declarations of variables and bodies of functions
- CPlus will create a“monkey.h.o” file which must be linked with the other “.o” files

Things to Look Out for *(cont.)*

- Conditional compilations–ex:

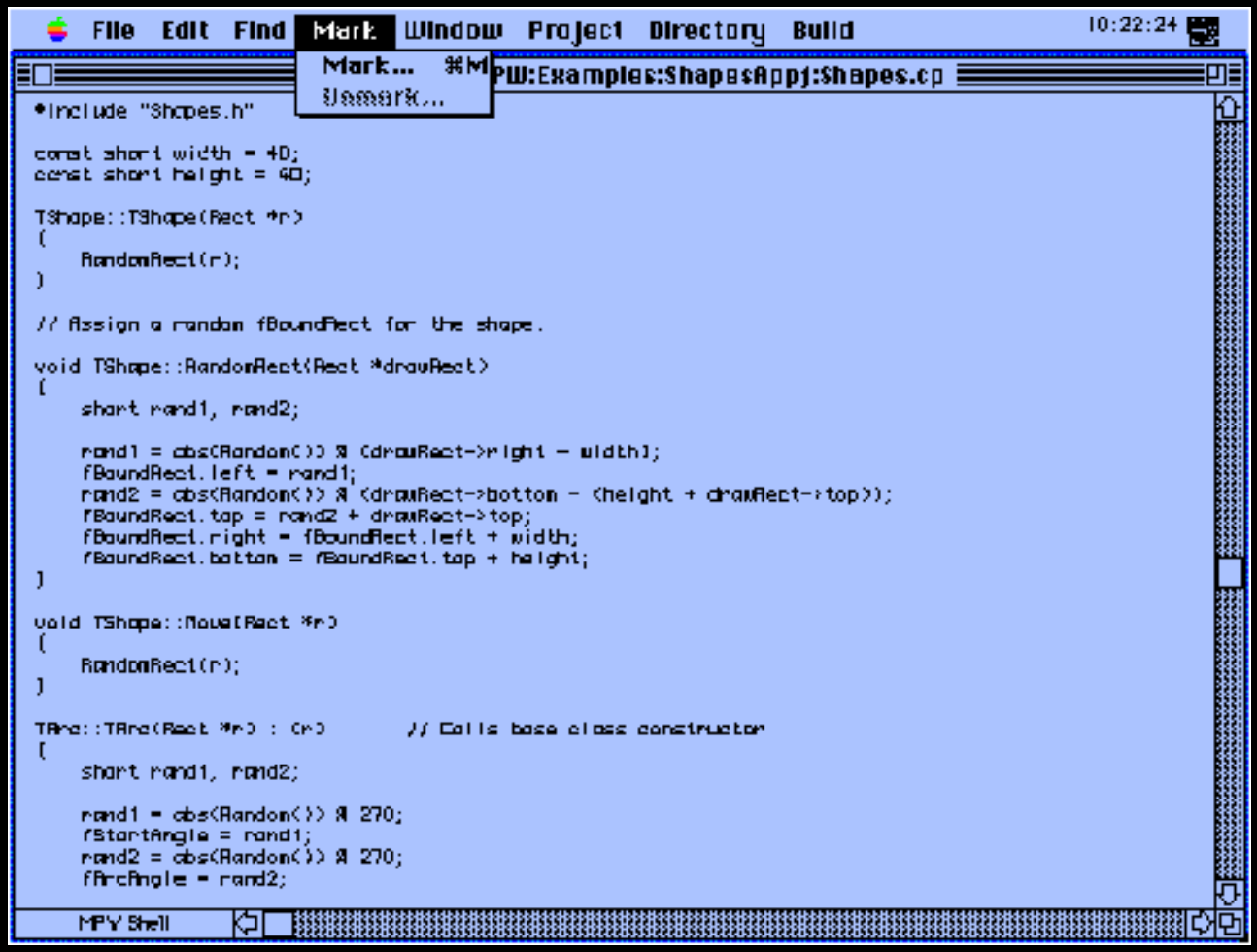
```
#ifdef MAC
#endif MAC
```

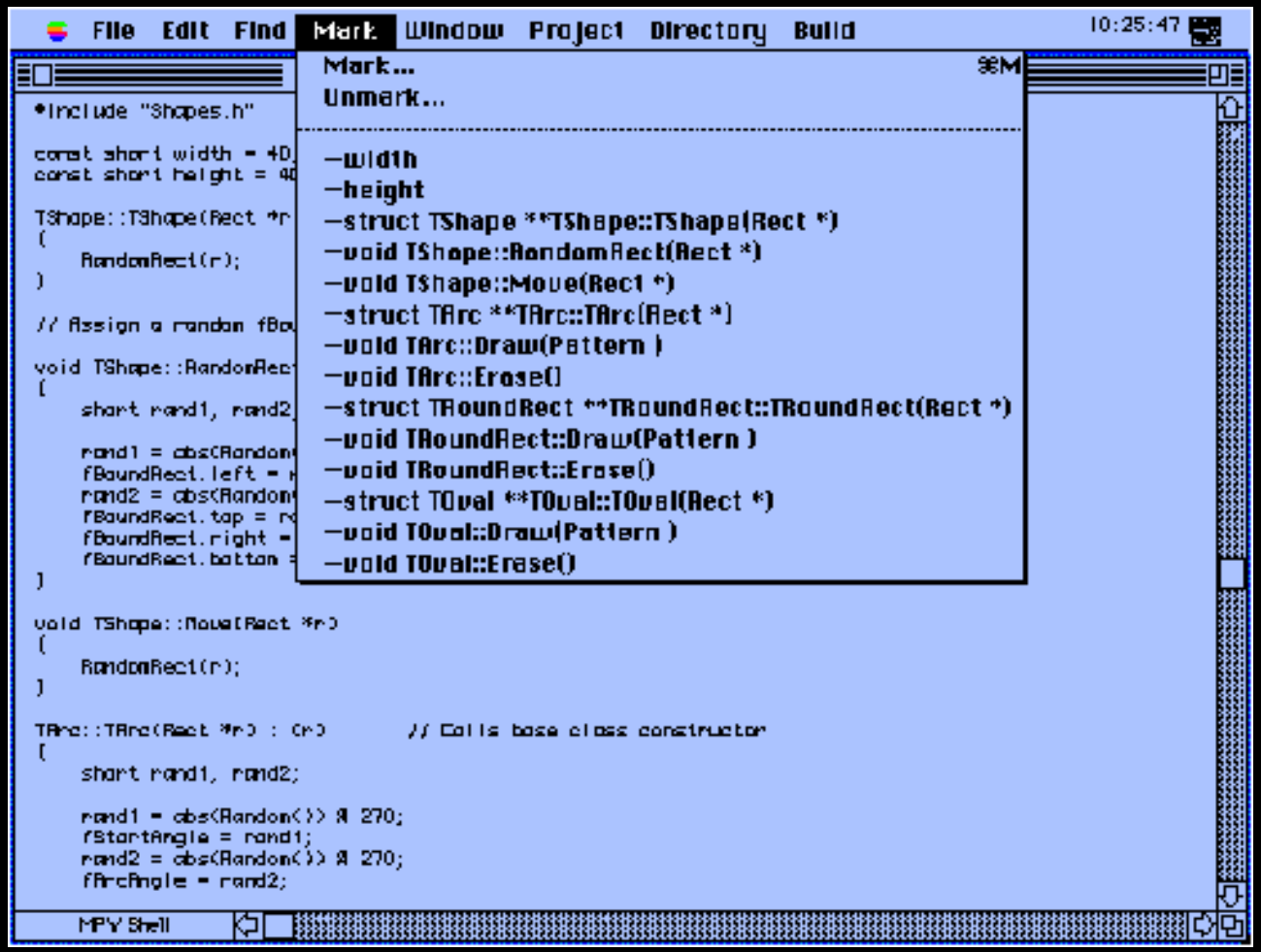
- The dump file ALSO remembers the preprocessor state. If MAC was defined for the dump, it will be defined for anything loaded from that dump. If you change macros for conditional compilation, you MUST rebuild the dump file!


Mark

What are “Marks?”

- Marks are MPW file markers stored in the resource portion of a file
- MPW C++ now will define marks (via “-mark option”)
- The options are:
  - mark fcts
  - mark types
  - mark data
  - mark all


File Edit Find Mark Window Project Directory Build
10:22:24
Mark... ⌘M
Unmark...
PW:Examples:ShapesAppj:Shapes.cp
#include "Shapes.h"
const short width = 40;
const short height = 40;
TShape::TShape(Rect *r)
{
RandomRect(r);
}
// Assign a random fBoundRect for the shape.
void TShape::RandomRect(Rect *drawRect)
{
short rand1, rand2;
rand1 = abs(Random()) % (drawRect->right - width);
fBoundRect.left = rand1;
rand2 = abs(Random()) % (drawRect->bottom - (height + drawRect->top));
fBoundRect.top = rand2 + drawRect->top;
fBoundRect.right = fBoundRect.left + width;
fBoundRect.bottom = fBoundRect.top + height;
}
void TShape::Move(Rect *r)
{
RandomRect(r);
}
TArc::TArc(Rect *r) : (r) // Calls base class constructor
{
short rand1, rand2;
rand1 = abs(Random()) % 270;
fStartAngle = rand1;
rand2 = abs(Random()) % 270;
fArcAngle = rand2;
MPW Shell


File Edit Find Mark Window Project Directory Build
10:25:47
Mark... ⌘M
Unmark...
—width
—height
—struct TShape **TShape::TShape(Rect *)
—void TShape::RandomRect(Rect *)
—void TShape::Move(Rect *)
—struct TArc **TArc::TArc(Rect *)
—void TArc::Draw(Pattern)
—void TArc::Erase()
—struct TRoundRect **TRoundRect::TRoundRect(Rect *)
—void TRoundRect::Draw(Pattern)
—void TRoundRect::Erase()
—struct TOval **TOval::TOval(Rect *)
—void TOval::Draw(Pattern)
—void TOval::Erase()
#include "Shapes.h"
const short width = 40
const short height = 40
TShape::TShape(Rect *r
{
RandomRect(r);
}
// Assign a random fBo
void TShape::RandomRect
{
short rand1, rand2;
rand1 = abs(Random
fBoundRect.left =
rand2 = abs(Random
fBoundRect.top = r
fBoundRect.right =
fBoundRect.bottom =
}
void TShape::Move(Rect *r)
{
RandomRect(r);
}
TArc::TArc(Rect *r) : Cr) // Calls base class constructor
{
short rand1, rand2;
rand1 = abs(Random()) % 270;
fStartAngle = rand1;
rand2 = abs(Random()) % 270;
fArcAngle = rand2;
MPW Shell

MultiFinder Option

- Lets CPlus use MultiFinder memory (via “-mf” flag)
- Allows MPW Shell’s partition to be kept small

Jack Palevich

Adv. Tech. Group
C++/MacApp Pioneer

Using C++ with MacApp

Advantages of C++ vs. Object Pascal

- Private and protected members
- Function and operator overloading
- Static, stack, and pointer based objects
- Inline methods and functions
- Multiple inheritance

Advantages of Object Pascal vs. C++

- Development cycle is faster
- More error checking ({$R+} and {$H+})
- WITH statement
- Nested procedures
- OVERRIDE keyword
- Does not require manual specification of virtual functions

Emulating Object Pascal Features

- WITH via temporary pointers
- Exception handling via macros
- Nested Procedure Pointers via glue code (Tech Note 265)

PascalObject Compatible Features

- Public, protected and private class members
- Inline methods for speed
- Const arguments keep you honest
- Pass by reference for speed and clarity
- Operator overloading
- Function overloading

PascalObject Incompatible Features

- Use pointer, static, and automatic storage
- Override new and delete
- Use multiple inheritance

Macintosh Toolbox Data Structures

- Can’t use virtual methods without adding vtable pointer
- Simplify parameter-block interfaces
- QuickDraw data structures
  - Points, Rects, Regions are naturals!
  - Use operator overloading +, -, =, etc.
  - Iterators for Resource files

Macintosh Toolbox Data Structures

- Network objects
  - Constructors open ports
  - Destructors close ports

Questions & Answers

The power to be your best