ANSI C Changes

Jonathan Hoyle

Eastman Kodak

10/5/00

ANSI C Changes

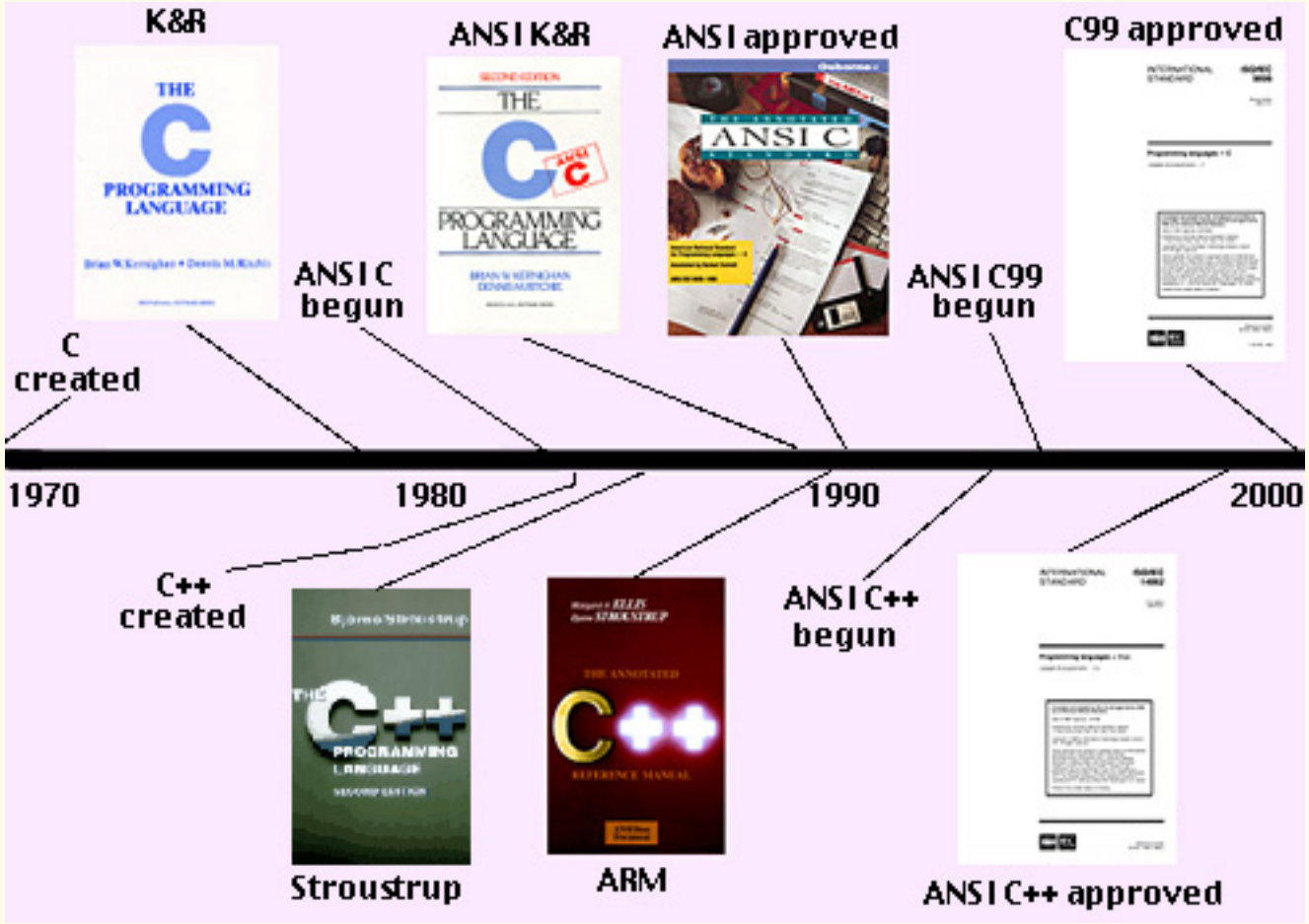
- ✓ Introduction
- ✓ Changes to C in conformance to C++
- ✓ New additions to C “friendly” to C++
- ✓ New additions to C “unfriendly” to C++
- ✓ What has not changed in C
- ✓ Conclusion

Introduction

Timeline

- ✓ 1969 - Ken Thompson creates Unix, B from BCPL
- ✓ 1970 - Thompson & Ritchie evolve B to C
- ✓ 1978 - K&R's "The C Programming Language"
- ✓ 1982 - ANSI forms a committee on standardizing C
- ✓ 1988 - K&R updated for ANSI Draft
- ✓ 1989 - ANSI approves C (called "C89")
- ✓ 1990 - ISO approves C (called "C90")
- ✓ 1995 - New committee formed for "C9X"
- ✓ 1999 - ISO approval (called "C99")
- ✓ 2000 - ANSI approves "C99"

Timeline


K&R
ANSI K&R
ANSI approved
C99 approved
THE C PROGRAMMING LANGUAGE
THE C PROGRAMMING LANGUAGE
ANSI C
ANSI C begun
ANSI C99 begun
C created
1970
1980
1990
2000
C++ created
ANSI C++ begun
THE C++ PROGRAMMING LANGUAGE
Stroustrup
THE ANNOTATED C++ REFERENCE MANUAL
ARM
ANSI C++ approved

Why Should We Care About C?

- ✓ Because I brought doughnuts.
- ✓ C is a subset of C++
- ✓ ANSI C++ compilers are usually ANSI C
- ✓ We may inherit C code
- ✓ We may still have to support C's public interface (libraries, etc.)
- ✓ There are some very useful features in C99 which we may want to use even in C++.

Changes to C in Conformance with C++

C++ style comments

```
/* Old style comments */
DoSomeStuff();

// New style comments now in C!
DoSomeMoreStuff();
```

☐ C90 ☑ C99 ☑ C++

No implicit int and prototypes

- Implicit int variables and function prototypes no longer supported.

```
const x = 0; //C90: OK; C99,C++: Error

void foo()
{
    // No implicit "int bar()" prototype
    x = bar(); //C90: OK; C99,C++: Error
}
```

☐ C90 ☑ C99 ☑ C++

Intermixed declarations & statements

```
void foo()
{
    // Part 1
    int a, b, c;
    PerformStuff(a, b, c);

    // Part 2
    int x, y, z;
    PerformStuff(x, y, z);
}
```

 C90

 C99

 C++

Conditional Expression Declarations

```
for (int n = 0; n < 10; n++)
{
    DoLoop(n);
}
```

☐ C90 ☑ C99 ☑ C++

Conditional Expression Declarations

```
int  n = 100;

for (int n = 0; n < 10; n++)
{
    DoLoop(n);
}

printf("%d", n);   // Prints 100
```

☐ C90 ☑ C99 ☑ C++

Inline functions

```
inline int Clamp(int inParm)
{ return (inParm < 0) ? 0 : inParm; }

void foo()
{
    …
    a = Clamp(x);
    b = Clamp(y);
    c = Clamp(z);
}
```

☐ C90 ☑ C99 ☑ C++

bool type

- ANSI C adds the _Bool keyword and type
- It is an integer type holding (at least) 1 bit
- It is an arithmetic type
- <stdbool.h> includes the definitions:
  - `#define bool  _Bool`
  - `#define true  (_Bool) 1`
  - `#define false  (_Bool) 0`
- C++ will provide an empty <stdbool.h>

☐ C90 ☑ C99 ☑ C++

Alternate punctuation tokens

- Requires the <iso646.h> header file (already supplied by C++):

and	&&	not_eq	!=
and_eq	&=	or	||
bitand	&	or_eq	|
=			
bitor	|	xor	^
compl	~	xor_eq	^=
not	!		

☐ C90 ☑ C99 ☑ C++

Digraph Punctuation Tokens

- Two character token replacements:

`<:`	`[`	`:>`	`]`
`<%`	`{`	`%>`	`}`
`%:`	`#`	`%:%:`	`##`

```
// Function definition using digraphs:
%:include <string.h>

void PCopy(char d<::>,const char s<::>)
<% strncpy(d, &s<:1:>, s<:0:>); %>
```

☐ C90 ☑ C99 ☑ C++

New Features “friendly” to C++

long long (64-bit integer type)

```
long long                x;
unsigned long long       y;

x = -9000000000000000000LL;
y = 18000000000000000000ULL;
```

☐ C90 ☑ C99 ☐ C++

#include <stdint.h>

```
// New types:
int8_t    int16_t    int32_t   int64_t
uint8_t   uint16_t   uint32_t  uint64_t

int_least n _t        uint_least n _t
int_fast n _t         uint_fast n _t

intmax_t              uintmax_t
intptr_t              uintptr_t
```

☐ C90 ☑ C99 ☐ C++

Variable Length Arrays

```
// Create a variable sized 2D matrix
void foo(int x, int y)
{
    double    theMatrix[x][y];

    ...
}
```

Predefined __func__ identifier

```
// __func__ returns function name
void foo()
{
    LogThis(__FILE__);  // “main.c”
    LogThis(__LINE__);  // 27
    LogThis(__func__);  // “foo”
    ...
}
```

☐ C90 ☑ C99 ☐ C++

Variable Argument Macros

```
#ifdef DEBUG
    #define DPr(...) printf(__VA_ARGS__)
#else
    #define DPr(...)
#endif //DEBUG

DPr("Hello, World!");
DPr("Error = %d", 10);
DPr("%d\t%5.3f\t", 1999, 3.14159);
```

☐ C90 ☑ C99 ☐ C++

Designated Initializers

```
int  a[5] = { 2, 3, 5, 7, 11 };
int  a[10] = { [0] = 100, [3] = 200 };

struct B
{
    int         x;
    double      y;
};

struct B  b = { .x = 3, .y = 2.71828 };
```

☐ C90 ☑ C99 ☐ C++

restrict-qualified pointers

- ✓ ANSI C adds the new keyword restrict
- ✓ Restricted ptr's have sole access to data
- ✓ Designate non-overlapping pointers
- ✓ Allows for compiler optimization
- ✓ Used for:
  - function prototypes
  - fields in struct definitions
  - return types for memory allocation

☐ C90 ☑ C99 ☐ C++

restrict-qualified pointers

```
void AddFloat(float *restrict sumPtr,
                      const float *ptr1,
                      const float *ptr2,
                           int numItems)
{
    for (int i = 0; i < numItems; i++)
         sumPtr[i] = ptr1[i] + ptr2[i];
}
// sumPtr should not overlap ptr1, ptr2
```

☐ C90 ☑ C99 ☐ C++

Other sundry changes...

- ✓ Flexible array members:
  - struct S
    {
        int count;   // fixed member
        int   myArray[]; // flexible member
    };
- ✓ Compound literals:
  - char *String = “Hello, World!”

☐ C90 ☑ C99 ☐ C++

Other sundry changes...

- ✓ Hexadecimal floating point literals:
  - `float pi = 0x3.243F6A88;`
- ✓ New _Pragma operator:
  - `_Pragma(`*directive*`)` `// #pragma` *directive*
- ✓ Empty Macro arguments:
  - `#define Add(x,y,z) (x + y + z)`
  - `Add(,,1);` `// becomes: ( + + 1);`
- ✓ enum list with trailing comma:
  - `enum T { t0, t1, t2, };`

☐ C90 ☑ C99 ☐ C++

New Features “unfriendly” to C++

complex arithmetic type

- ANSI C adds _Complex and _Imaginary
- <complex.h> includes the definitions:
  - `#define complex` `_Complex`
  - `#define imaginary` `_Imaginary`
  - `#define I` *imaginary identity*
- ANSI C++'s templated complex<> type is incompatible with ANSI C's complex
- C++'s complex.h header file conflicts with ANSI C's of the same name.

complex arithmetic type

```
// ANSI C complex function
double complex square(double complex z)
{
  return (z * z);
}
// ANSI C++ complex function
complex<double> square(complex<double> z)
{
  return (z * z);
}
```

complex arithmetic type

```
// Suggested solution
#include <complex.h>

#ifdef __cplusplus
  typedef complex<float> complex_float;
  typedef complex<double>complex_double;
#else
  typedef float complex  complex_float;
  typedef double complex complex_double;
#endif //__cplusplus
```

C/C++ clog identifier conflict

- ANSI C defines clog() in <complex.h> as the complex logarithm function.
- ANSI C++ already defined clog in <iostream.h> as standard error logging.

```
// Possible conflict in clog usage
void foo()
{
    clog << clog(2.71828) << endl;
}
```

Type-generic math functions

- Math functions (formerly defined with double's) have a "dynamic" return type.
- Affects only functions defined in <math.h>, such as sin(), cos(), sqrt(), etc.
  - `sin(3.14159);  //returns a double`
  - `sin(3.14159F);  //returns a float`
  - `sin(3.14159L);  //returns long double`
- Macro-based (requires <tgmath.h>)
- May conflict with C++'s overloading rules

What has not changed in ANSI C

Unchanged in C

- ✓ K&R-style prototypes still legal (deprecated):

```
void foo(a, b)
     int  a;
     int  b;
{ ... }
```

- ✓ goto's still not deprecated
- ✓ sizeof ('a') == sizeof(int), not sizeof(char)
- ✓ "Hello" is still a char *, not a const char *

☑ C90 ☑ C99 ☐ C++

Unchanged in C

- ✓ Duplicate typedef's still illegal in C
- ✓ enum constants are still signed int's
- ✓ Tentative definitions still legal in C:

```
int  x;          // tentative defn
int  x = 1;      // explicit defn
```

- ✓ String initializers without null-termination:

```
char s[6] = "Hello"; // legal C/C++
char s[5] = "Hello"; // legal C only
```

☑ C90 ☑ C99 ☐ C++

Conclusion

Conclusion

- ✓ Best features of C99:
  - Variable length arrays
  - Variable argument macros
  - New integer types
  - restrict-qualified pointers
- ✓ Worst features of C99:
  - incompatible features with C++
  - K&R prototypes still legal
  - goto's not deprecated

Resources

- ANSI/ISO C 1999 Specification:
  - http://www.jonhoyle.com/ANSI/c.pdf
- ANSI/ISO C++ 1998 Specification:
  - http://www.jonhoyle.com/ANSI/cplusplus.pdf
- ISO C99 Rationale:
  - http://www.jonhoyle.com/ANSI/cplusplus.pdf
- Incompatibilities between ISO C and ISO C ++:
  - http://home.flash.net/~dtribble/text/cdiffs.htm

Q & A