C++ For C Programmers
Week 1 Part 1

- Overview and Course Organization
- C++ as a Better C
- Problem 1 – Convert C Code – to be posted

Course work

1. Homework – 3 Programs to Code
2. One Final- closed book

C++ is Better than C

- Better?
- Mays is better than Mantle
- Butter is Better than Margarine
- C++ is better than Java

Quiz: Is margarine better?

- In what sense is margarine better than butter?

Margarine v. Butter

- Calories in a pat 35 calories – a tie
- Some Margarine is healthier
- Some Margarine contains no cholesterol
- Butter is made from animal fat
- Chefs like butter
- Butter has more vitamins
- Bottom Line _____

History

Algol	• International Group
BCPL	• Martin Richards
B	• Ken Thomson
Traditional C	• Dennis Ritchie
K&RC	• Kernighan & Ritchie
ANSI C	• ANSI Committee
ANSI / ISO C	• ISO Committee
C99	• Standard Committee

C vs C++ Language

- C Ritchie 1972 –SIL, small 29 keywords
- C++ Stroustrup 1985, large 63 keywords by 1996
- C supports imperative programming
- C++ “Swiss Army Knife” also supports OO and generic programming

Quiz

- C++ is better than C because:
  A) It was invented in 1982-85 vs 1972
  B) Bjorn Stroustrup is a better language designer than Dennis Ritchie
  C) It has more features than C
  D) It is simpler than C

Answers Quiz 1

- C in 1972 v. C++ 1982-85 is true but does not guarantee “better”
- Both Dennis and Bjarne are excellent – Dennis won the Turing Prize
- Best answer – C is a subset and C++ has better libraries
- False – C++ is far harder
- Bottom Line_____

Java vs C++

- Java more of a purely OO language
- Java- Sun 1995 “Smalltalk with a C syntax”. Java has single root object
- Java has the JVM – universal semantics
- C++ originally compiled to C
- C++ no hidden overhead
- Ergo, no GC and default not virtual fcn call

If you don't know Java or OO

- Java is not critical to this class – but many of you have this background so I will be making some comparisons

- Object-Orientation knowledge is a "prerequisite" – so if you need review read the wikipedia article on OO.

Quiz - Object

- Which of these ideas best fit OO?

a) Control flow should use recursion.
b) Data and operations are bound together.
c) Type safety is critical.
d) Automatic GC is available.

Answers

- Object Orientation is not about flow of control-but you should be comfortable with recursion.
- Best answer: An object is a data type and its operations that can be either native or user defined.
- Type safety is usually a property of a good OO language.
- Garbage Collection can be built-in to an OO language or provided for a user-defined object.

Example Program:Shooting Craps

C Program- Dice Probability

- #include <stdio.h>
- #include <stdlib.h>
- #include <time.h>
- #define SIDES 6
- #define R_SIDE (rand() % SIDES + 1)

C Program

- int main(void)
- {
- int n_dice = 2;
- int d1, d2, trials, j;
- int* outcomes =
- calloc (sizeof(int), n_dice * SIDES +1);

C Program

```
srand(clock());
printf ("\nEnter number of trials: ");
scanf("%d", &trials);
for (j = 0; j < trials; ++j)
   outcomes[(d1 = R_SIDE) +( d2 = R_SIDE)]++;
printf( "probability\n");
for (j = 2; j < n_dice * SIDES + 1; ++j)
   printf(  "j = %d  p = %lf\n" , j ,
     (double)(outcomes[j])/trials);
}
```

C

- libraries
- #define macros
- main()

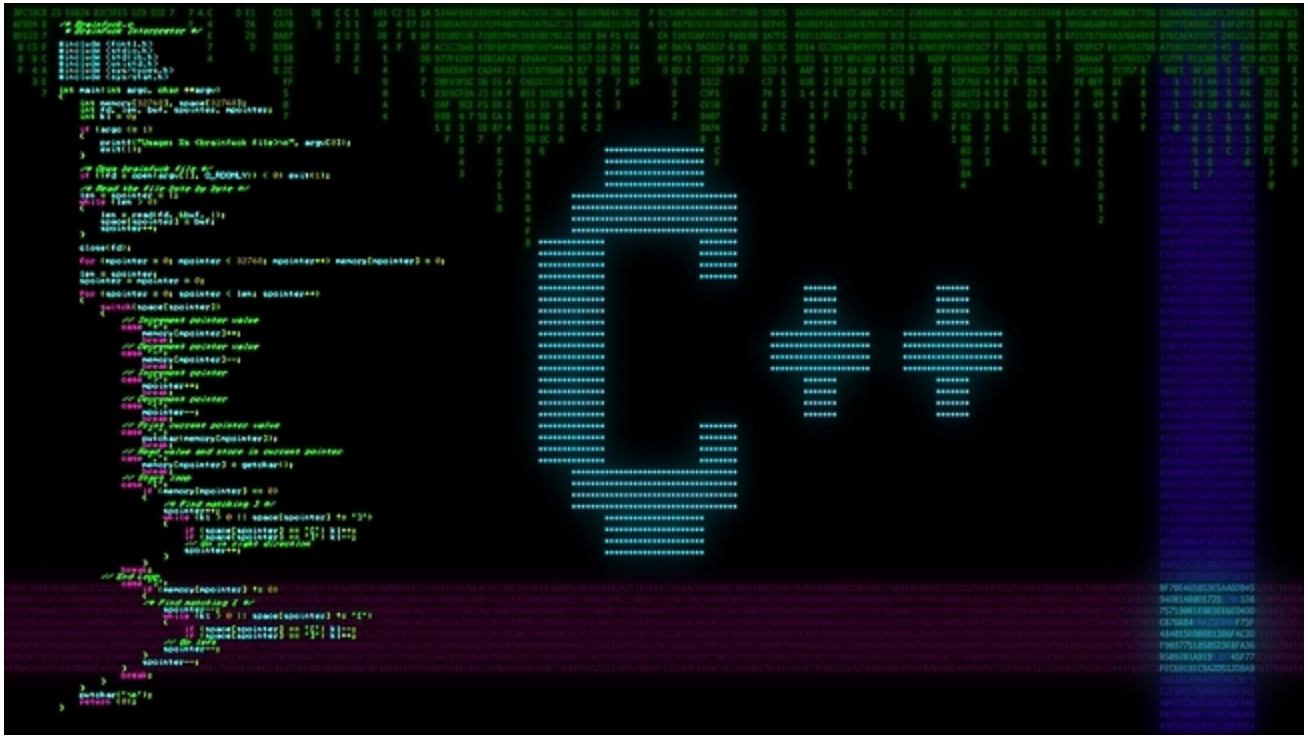
Quiz

- True or False:
- main() is the function in C where programs start executing.
- printf() can be used for input.
- int is more precise than float.

Answers

- main() is the function in C where programs start executing. True
- printf() can be used for input. False . In stdio library scanf() is the analogous input function
- int is more precise than float. True. On most machines int can represent +/- 2 billion integer values. Float usually has 6 significant figures of accuracy when representing integers

C++ Version



C++ version

- //The following program computes
- //the probability for dice possibilities
- //
- #include <iostream>   //drops .h still available
- #include <cstdlib>
- #include <ctime>
- using namespace std;
- const int sides = 6; //replaces many sharp defines
- inline int r_sides(){ return (rand() % sides + 1); }

C++ Improvements

- NB some of these are adopted in modern C
- Different libraries most old C libraries still available – usually as c….
- Inline - replaces code macro – benefit?
- Rest of line comments //

C++ Program

```
•int main(void)
•{
•   const int   n_dice = 2;
•   int  d1, d2;
•   srand(clock());   //why?
•   cout << "\nEnter number of trials: ";
•   int trials;
•   cin >> trials;  //compare to scanf
•   int* outcomes = new int[n_dice * sides +1];   TYPE* s;
•
```

Quiz

- srand(clock()); //why?
-

Answer

- srand() –initializes the random number sequence
- Don’t want the same results each time in a simulation
- clock() Returns the processor time consumed by the program. The value returned is expressed in **clock** ticks .
- Fine grained and different when main() starts running

C++ Advantages

- Iostream cin>> type safe
- Declaration can be intermixed after executable
- Use of new

C++

```
for (int j = 0; j < trials; ++j)
    outcomes[(d1 = r_sides()) +( d2 = r_sides())]++;
  cout << "probability\n";
  for (int j = 2; j < n_dice * sides + 1; ++j)
    cout << "j = " << j << " p = "
         << static_cast<double>(outcomes[j])/trials
         << endl;
}
```

C++ Advantages

- Safe casts
- *for statement* can include declaration initializer
- *endl* iomanipulator can be placed in an ostream

C++

- Bjarne Stroustrup: More to the point, I don't think "purity" is a virtue. The signal strength of C++ is exactly that it supports several effective styles of programming (several paradigms, if you must), and combinations of these styles. Often, the most elegant, most efficient, and the most maintainable solution involves more than one style (paradigm).

C++ : so far

- More type safe than C with more elegance

ADDENDUM HW1

```
#include <stdio.h>
#include <math.h>

double fcn(double x)
{ return x * x - 1; }

double fcn2(double x)
{ return x * x * x * x * x - exp( x ) -2; }

double root_find(
   double f(double),   /* function with root */
   double a, double b, /* interval (a, b) with root */
   double eps,         /* accuracy */
   double *root        /* root --output*/
 )
{

    if ( b - a  < eps){
       *root = (b + a)/2.0;
       return f(*root);
    }
    else
    if ( f(a)* f( (b + a)/2) <= 0)
       return root_find(f, a, (b+a)/2, eps, root);
    else
       return root_find(f, (b+a)/2, b, eps, root);
}

int main(void)
{
   double x;

   printf("Root Finder\n");
   root_find(fcn, 0.0,  4.0, 0.00001, &x);
   printf("root is at %g with residual %g \n", x, fcn(x));
   root_find(fcn2, 0.0,  14.0, 0.00001, &x);
   printf("root is at %g\n", x);
   return 0;
}
```

Rewrite as C++

- Rewrite in Modern C++
- Properly comment changes made
- Grades: Correct answer for running code
- Peer score:

Homework 1

- If you are using the text
- C++ for C Programmers: 3rd edition,
  - Pohl ISBN 0-201-39519-3 read ch1 and 2
- By next week –, turn the equivalent C++ program
- See assignment 1 write-up for more details.

C++ for C Programmers
Week 1 Part 2

- C++ as a Better C - continued

C++ is Better than C

- More Type Safe
- More Libraries
- Less reliance on preprocessor
- OO vs imperative

So far

- inline, const
- static_cast<*type*>
- namespaces
- iostream
- Declarations anywhere including
  - for-statement initialization

Quiz T/F

- const double PI = 3.14159;
- A) creates a non-mutable variable PI
- B) is equivalent to #define PI 3.14159
- C) both are true

Answers

- const double PI = 3.14159;

- A) creates a non-mutable variable PI
  - True, const means the initialized variable PI cannot have its value changed in the scope of this declaration.

- B) is equivalent to #define PI 3.14159
  - False, PI here is a macro – and is textually substituted for within the scope of the #define which is normally file scope. This is old C style and should be avoided.
- C) both are true -False

Swap in C

- void swap(int* i, int* j){
  int temp = *i;
  *i = *j;
  *j = temp;
  }
  Typical C idiom for passing parameters that need modification.

C-Swap

- ```
  void swap_double(double* i, double* j){
    double temp = *i;
    *i = *j;
    *j = temp;
  }
  ```

- In C each function in a given scope most have a unique name (no function overloading).
  ```

C Program

```
int main()
{
   int m = 5, n =10;
   double x = 5.3, y =10.6;
   printf("inputs: %d, %d\n", m, n);
   swap(&m, &n);
   printf("outputs: %d, %d\n", m, n);
   printf("double inputs: %lf, %lf\n", x, y);
   swap_double(&x, &y);
   printf("double outputs: %lf, %lf\n", x, y);

}
```

Draw Boxes for what happens

- M: [5] , N: [10]
  - Swap called with address of M and N &
- M: [10] , N: [5]

C Summary

- Call by reference simulated with pointers

C++ Version

- C++ has call by reference argument passing

C++ Program

- ```
  #include <iostream>
  using namespace std;

  inline void swap(int& i, int& j){
    int temp = i;
    i = j;
    j = temp;
  }

  inline void swap(double& i, double& j){
    double temp = i;
    i = j;
    j = temp;
  }
  ```
  C++ has overloading and call by reference.
  ```

C++

- ```
  int main()
  {
     int m = 5, n =10;
     double x = 5.3, y =10.6;
     cout << "inputs: " << m << " , " << n << endl;
     swap(m, n);
     cout << "outputs: " << m << " , " << n << endl;
     cout << " double inputs: " << x << " , "
        << y << endl;
     swap(x, y);
     cout << " double outputs: " << x << " , "
        << y << endl;
  }
  ```
  ```

C++ Overloading

- "signature matching algorithm".
- The two routines need different types or numbers of parameters.
- integer swap() is (int&, int&) double swap() is (double&, double&).

Why C++ is better

- Having the same name for conceptually the same activity promotes readable code.

- Overloading based on signature and later the use of generics is a powerful reuse mechanism.

Quiz

- std::cout is found in what header file?
- inline can be used when declaring____?
- What is the difference between long and long long and int?

Answers

- std::cout is found in <iostream>
- inline can be used when declaring functions to speed them up.
- What is the difference between long and long long and int? C++ allows for longer integer types starting with a shortest type short.

Generics in C++

"Generics"

Programming using templates

Generics in C++

“Generics”

Programming using templates

Influenced by Alex Stepanov

C++ template

• Generic programming in C++ was strongly influenced by Alex Stepanov.

• Much code is the same except for type. If the code can be applied universally there can be a big saving in coding and debugging.

• Stepanov while at HP in the late 80's developed a set of generic libraries using a construct called a template. STL is largely based on this work.

C++

- template <class T>

```
inline void swap(T& d, T& s)
{
        T temp = d;
        d = s;
        s = temp;
}
```

C++Template function

- Prescription: Take a normal function and
  - Add template <class T>   //T is id

  - Now for int or double or whatever use T when called as in swap(a, b);

  - The compiler uses a signature matching routine to infer what the code should use for type.

C++ use of template

```
• int main()
{
   int m = 5, n =10;
   double x = 5.3, y =10.6;
   complex<double> r(2.4, 3.5), s(3.4, 6.7);
   cout << "inputs: " << m << " , " << n << endl;
   swap(m, n);
   cout << "outputs: " << m << " , " << n << endl;
```

C++ template

```
• cout << "outputs: " << m << " , " << n << endl;
   cout << " double inputs: " << x << " , "
                << y << endl;
   swap(x, y);
   cout << " double outputs: " << x << " , "
                << y << endl;
   cout << " complex inputs: " << r << " , "
                << s << endl;
   swap(r, s);
   cout << " double outputs: " << r << " , "
                << s << endl;
}
```

Compile generics to instances

- The compiler uses the template to write code appropriate to each set of parameters.
- The m,n parameters are ints - so the compiled code is swap(int&, int&) signature.
- Similarly, for the other two signatures. So the compiler is compiling code appropriate to each distinct signature.

Exercise

- Take what you know about summing ‘n’ numbers (using a C array) and write that code.
- Debug in C++
- Change to a template

Answer

- Template<class summable>
- summable sum<summable> (data[], int size){
- summable asum = 0;
- for(int i = 0; i < size; ++i)
- asum += data[i];
- return asum;
- }

- Run and discuss