CS11600-1
Intro to Programming II

Announcement:

- **Newsgroup uchi.cs.cs11600 has been created.**

Last Time

- **Recursivity (Section 3.1)**
- **Additional discussions**
- **More about Pointers (Section 2.1)**

Last Time

- Not specific about
  - Compilers & C/C++ & Out-of-order execution
  - Language level vs. Compiler level

Today

- Pointers & References. Function Arguments (Section 3.1, 3.3)
- Namespaces (Section 3.6)
- Next time
  - I/O Operations (Examples)
  - String Operations (Examples)
  - Dynamic Allocation of Memory (Section 3.7)

Pointer Type

- Holds the address of another object, allowing for indirect manipulation of that object
- Has an associated type (void ...)

Example:

```
int a = 6, *ptr_a = &a, **pptr_a = &ptr_a;
cout << a << *ptr_a << endl;
**pptr_a = 7;
cout << a << *ptr_a << endl;
cout << &a << endl;
```

- const and pointers:

```
const int* a; // pointer to a const object
int *const a; // constant pointer
```

Arrays and Pointers

C and C++:

```
int vec[4] = { 1, 2, 3, 4 };
vec  == &vec[0]
*vec = vec[0]
vec + 1 = &vec[1]
*(vec + 1) = vec[1]
```

Note for C++:

```
#include <vector>
vector <int> ivec(10);
ivec[1] = ivec[2];
if (ivec.empty ()) { ... }
```

Example (e9.cc)

```
#include <iostream>

void ia_print (int* pbegin, int* pend)
{
    while (pbegin != pend)
        { cout << *pbegin << " ' "; ++pbegin; }
}

int main ()
{
    int ia[4] = { 0, 1, 2, 3 };
    ia_print (ia, ia+4);
}
```

Pointers to Functions

```
int (*pf) (int, int);    // (!= int *pf (int a, int b);  )

int max (int a, int b) { ... }
int min (int a, int b) { ... }

// int (*apf[2]) (int, int) = { min, max };

int main ()
{
    if (<cond>)
        pf = min;
    else
        pf = max;
   // Other operations
   pf (3, 4)
}
```

Example (e10.cc)

```
#include <iostream>

int min (int*, int);
int (*pf) (int*, int) = min;

const int size = 5;
int ia[size] = { 7, 8, 9, 2, 5};

int main () {
    cout << min (ia, size) << pf (ia, size) << endl;
    return 0;
}
int min (int* ia, int sz) {
    int minVal = ia[0];
    for (int ix = 1; ix < sz; ++ix)
        if (minVal > ia[ix])
            minVal = ia[ix];
    return minVal;
}
```

Reference Types

- Introduce an alias, an alternative name for an object (?)
- Allow indirect manipulation without the necessity of a pointer

Example:

```
int a = 6, &ref_a = a;
cout << a << ref_a << endl;
```

- Note: once created a reference cannot be changed

Function Support

- **Argument Passing**
- **Returning Value**

Argument Passing

```
inline int
max (int a, int b)
{
  if (a > b)
    return a;
  else
    return b;
}

int
main ()
{
  cout << max (10, 20) << endl;
  return 0;
}
```

Argument Passing

```
inline  void
swap (int &a, int &b)
{
   int tmp = a;
   a = b;
   b = tmp;
}

int
main ()
{
   swap (int a = 10, int b = 20);
   return 0;
}
```

Array Passing

```
void
reverse (int *arr, int size)
{
  for (int i = 0; i < size; i++)        // Logic error ...
     swap (arr[i], arr[size-i-1]);
}
```

== int reverse (int arr[], int size) {... }

== int reverse (int arr[20], int size) { ... }

A special case:

C strings = arrays with the last element 0 ('\0' and not '0')

Array Passing

```
void
reverse (int (&arr)[20])
{
 for (int i = 0; i < 10; i++)
   swap (arr[i], arr[19-i]);
}
```

!= int reverse (int *arr, int size) {... }

Array Passing

```
int
fileHandle (char *name = "input.txt")
{
  istream input (name);
  if (!input) return -1;
  /* Other computations */
  input.close ();
  return 0;
}
```

Other examples:

- create objects with default initial values / sizes
- screen management -> create new windows with default layouts if not otherwise specified

Ellipsis

```
void debug (char *fmt, ...)
{
  va_list vl; int d; char c, *p, *s;
  va_start(ap, fmt);
  while (*fmt)
     switch(*fmt++) {
       case 's':        /* string */
            s = va_arg(ap, char *);
            cout << " string: " << s;
            break;
       case 'd':        /* int */
            d = va_arg(ap, int);
            cout << " int: " <<  d;
            break;
     }
   cout << endl; va_end(ap);
}
```

Return Value

main (): Handling Line Opt

./a.out -d -f input.txt

int main () { ... }

int main (int argn) { ... }

int main (int argn, char *argv[]) { ... }

argv[0] = "./a.out"
argv[1] = "-d"
argv[2] = "-f"
argv[3] = "input.txt"

Namespaces

- Block of definitions delimited by { } and associated with a name
- Elements
  - Namespace Definition
  - The Scope Operator (::)
  - Member Definitions
  - Unnamed Namespaces
  - Namespace Members
  - The standard namespace (*std*)

Example

```
namespace A {

 int swap (int &a, int &b) { ... }

}

int b = A::swap (10, 20);
```

or

```
using namespace A;
```

Example - Definition

```
// math.hh

#ifndef _MATH_HH_
#define _MATH_HH_

namespace math_space {

   const double PI = 3.1415;

   double pow (double, double);
   double log (double, double);

}

#endif // _MATH_H_
```

Example - Member Def.

```
// math.cc

#include <math.h>
#include "math.hh"

double
math_space::pow (double a, double b)
{
  return ::pow (a, b);
}

double
math_space::log (double a, double b)
{
  return ::log (b) / ::log (a);
}
```

Example - Scope Operator

```
// main.cc

#include <iostream>
using namespace std;
#include "math.hh"

int
main (int argn, char *argv[])
{

   cout  << math_space::PI << " " <<
            math_space::log (10, 100) << " " <<
            math_space::log (100, 2) << endl;

   return 0;

}
```

Observations

- **Namespace definitions can be discontiguous**
- **Can span multiple files**
- ***Previous example already used these two features***

Example - Unamed NM

```
namespace {

 int swap (int &a, int &b) { a += b; b = a - b; a -= b; }

}
```

Anywhere inside *some_file.cc*

```
 swap (10, 20);
```

is a correct call to *swap*.

Example - *using*

```
#include "math.hh"

using namespace math_space;

int
main (int argc, char *argv[])
{
 cout << PI << " " << log (10, 1000) << " " << pow (10, 3) << endl;
 return 0;
}
```

Example - *std*

- All components of the C++ standard library are declared/defined in namespace *std*

- Some compilers require *using namespace std;*

- or (a more restricted form)
  - *using std::string;*
  - *using std::cin;*

Homework

- **Due date: January 17, 8:00pm**

- Problem 0. Problem 7 / Page 81 (Navigating C++) -- nothing to hand in

- Problem 1. Write two functions (ten2tes and tes2ten) for translating large numbers represented as strings from base 10 to base 2, 8 or 16 and, respectively from 2, 8 or 16 to 10.

  char* ten2tes (char* bten, int base);

  char* tes2ten (char* tes);

  Example:

  10 (10) = xA (16) = 012 (8) = b1010 (2)

  20 (10) = x14 (16) = 024 (8) = b10100 (2)

- Problem 3. Problem 1 / Page 164 (Navigating C++)

I/O operations

Streams

- Output
  - <<
  - flush
  - setf, unsetf (ios_base::showbase, ios_base::oct)
  - *cout << 1234 << "," << hex << 1234 << "," << oct << 1234 << endl;*

- Input
  - >>
  - get ()
  - get (p, n, term)
  - getline (p, n, term)
  - setf, unsetf

Streams

```
#include <iostream>
#include <fstream>
#include <string>
int main (){
int counter[LETTER_COUNT];
   ifstream infile( "input.txt" );
   // Check if the open operation was succesful
   if (!infile)
        { cerr << "Error: unable to open input file!\n"; return -1; }
   // Initialize the array of counters
   for (int i = 0; i < LETTER_COUNT; i++) counter[i] = 0;
   // Read words one by one and count their content
   string word;
   while (infile >> word)
     for (int i = 0; i < word.size(); ++i)
       if ((word[i] - 'a' >= 0) && (word[i] - 'a' < LETTER_COUNT))
         counter[word[i] - 'a']++;
   // Print the result
   for (int i = 0; i < LETTER_COUNT; i++)
     cout << static_cast<char> ('a' + i) << "   " << counter[i] << endl;
   return 0;
}
```

FILE

```
#include <stdio.h>

int main (int argn, char *argv[])
{
 FILE *pf;
 if (pf = fopen (argv[1]))
  {
    char buffer[1024];
    do {
        int len = fgets (buffer, 1024, pf);
        write (stdout, buffer, len);
    } while (len > 0);
    fclose (pf);
}
```

String operations

- C++ style
- Inherited zero-terminated C string style

C++-style

string type (string):

- length == size
- empty
- []
- at
- +=
- =
- append
- insert
- replace
- find
- compare
- c_str

C++-style

```
#include <string>
int main ()
{
    char *p = "aaaaaaaaaaaaaaaaaa";

    string s0, s0 = "";

    string s1 = 'a'; // error
    string s2 = 7;  // error
    string s3 (7);  // error

    string s4 (7, 'a'), s5 = "Frodo", s6 = s5;

    string s7 (s5, 3, 2), s8 (p + 7, 3), s9 (p, 7, 3);

    s7 += s8; s7 += p;

    cout << s7[10] << " from: " << s7.c_str () << endl;
}
```

C-style

string.h:

```
...
 char *strcpy (char* p, const char* q);
 char *strcat (char* p, const char* q);

 char *strncpy (char* p, const char* q, int n);
 char* strncat (char*p, const char* q, int n);

 size_t strlen (const char* p);

 int strcmp (const char* p, const char* q);
 int strncmp (const char* p, const char* q, int maxlen);
...
```

C-style

```
int main () {

    char *p1 = "aaaaaaaaa", *p2 = "bbbbbbbbbb";

    strcmp (p1, p2);
    strncpy (p1, p2, 3);

}
```

Dynamic Memory

- **Lifetime of a *static* declared object is defined during execution**

- **Sometime is necessary to define objects with sizes and lifetime controlled by programmer (Ex: reading a string from stdin)**
  - ***not anymore a sustainable reason***

- **new / delete (C++)**
  - ***ptr = new Type <> delete ptr***
  - ***ptr = new Type (initial_value) <> delete ptr***
  - ***ptr = new Type[size1] ... [sizeN] <> delete []ptr***

- **malloc / free (C - lower level manipulation of memory)**
  - ***ptr = (type*) malloc (memory_size)***
  - ***free (ptr)***

Example (e11.cc)

```
#include <iostream>

int main ()
{
     while (1)
         new int[100];
    return 0;
}
```

Array Allocation

Example:

```
// A single object with value 1024
int *obj = new int (1024);
// An array
int *vec = new int [1024];
int (*mat)[1024] = new int [4] [1024];
delete[] mat;

const int *pci = new const int (1024);
```

Example (e12.cc)

```
#include <iostream>

int*[] alloc_mat (int dim1, int dim2) {
    int (*mat)[dim1] = new [dim1][dim2];
    return mat;
}

int main ()
{
    int (*mat1)[10] = alloc_mat (10, 10);
    int (*mat2)[20] = alloc_mat(20,20);
    delete []mat1;
    delete []mat2;
}
```

Type Conversion

- Implicit
  - int a;
  - a = 3.4;
- Required by Arithmetic Operations
  - 4.5 * 3;
- Explicit
  - (int) 3.4
  - static_cast<int> (3.4);
  - dynamic_cast (runtime casting)