CPS 100 - Program Design and Analysis I
Dr. Susan Rodger
Section: Review C++, Pointers
(handout)

Read Chap. 1, 6.4.

Design of Program

- design

problem
↓
algorithm
↓
program
↓
test? refine?

- Modular
  - functions - short, precise
  - pre-, postconditions
  - comments
  - Do not use global variables!

Review Input/Output

Input - Use cin

Output - Use cout

```
int score1, score2;
cin >> score1 >> score2;
cout << score1 << " " << score2 << endl;
```

DO NOT use printf, scanf!

Review strings

```
#include "CPstring.h"
```

Including the include statement above and using a proper Makefile will allow you to use the string class from CPS 6.

What can you do with a string?

```
string someword;
someword = "sillywilly";
someword = "dilly";
string word = "scrumptious";

cout << word[2] << word[7] << word[5] << endl;
cout << word.length() << endl;
word = word + someword;
word[20] = 'g';
```

Review Class

Create a class called Car

```
class Car
{
  public:
    Car();   // initially at position (2,2), direction east
    void MoveForward(int steps);
    void TurnLeft();
    void TurnRight();

  private:
     ?
 };
```

Write a program to create 2 cars, one moves to position (2,4) then both head west for 5 steps.

Review Vectors/arrays

A Vector is safer to use than an array.

Examples:

- `Vector <int> count(50);`
- `count.Fill(0);`
- `count[10] += 5;`
- `Vector <double> weight(100, 3.0);`
- `weight[34] = 136.3;`
- `Vector <string> words(20);`

- Resize a Vector

```
void ReadData(Vector <string> & words, int & curpos)
// precondition: Vector has room for at least one word
// postcondition: Vector has ‘‘curpos’’ words in it
{
 curpos = 0;
 while (cin >> words[curpos])
 {
   curpos += 1;
   if (curpos == words.Length())
   {
     words.Setsize(2*words.Length()+1);
   }
}
```

- `Vector <int> a(10,5), b(20,6);`

  `a=b;`

2-dimensional array - Use Matrix class

Struct

```
struct Member
{
   string Name;
   int Id;
};

Vector <Member> clublist(400);

clublist[7].Name = "Jackson";
clublist[7].Id = 1907;
```

Pointer

- indirect reference
- location in memory

```
Vector <Member *> club(5000);


club[0] = new Member;
club[0]->Name = "Rodger";
club[0]->Id = 6595;
```

Read in members and ids....

```
string name;
int idnumber;
while (cin >> name >> idnumber)
{


}
```

Dynamic Memory

1. new - get a chunk of memory from heap
2. delete - return memory to heap

Sort by Name and Sort by Id

```
Vector <Member *> clubNames(5000);
Vector <Member *> clubIds(5000);
```

What you can assign a pointer

1. new

   ```
   club[0] = new Member;
   int * x;
   x = new int;
   ```

2. & - address of another variable

   ```
   int a = 7;
   int * b = &a;
   *b = 3;
   cout << a << " " << b << " " << *b << endl;
   ```

3. value of another pointer

   ```
   Member * president;
   president  = club[8];
   ```

4. assign 0 or NULL

   ```
   Member * president = NULL;
   ```

Operators with pointers

- "->" - pointer is pointing to a struct, a field in the struct
- "&" - address of variable
- "*" - in declaration, means "pointer to" type
- "*" - with pointer, means "contents of"

Examples

```
string who = "me";
string * towho;   // dangling pointer
towho = NULL;
string * biff = &who;
*biff = "you";
cout << who << endl;

club[0]->Name = "Rodger";
(*club).Name = "Rodger";
```

Linked List

Data structure

- way to organize data in a list
- exact fit

```
struct NameNode
{
  string name;
  NameNode * next;
}
```

Build a list

```
string name;
NameNode * names = NULL;
NameNode * temp = 0;
while (cin >> name)
{
    temp = new NameNode;
    temp->name = name;
    temp->next = names;
    names = temp;
}
```

Easier, include a constructor

```
struct NameNode
{
  string name;
  NameNode * next;

  NameNode(string newname, NameNode * ptr)
  {
    name = newname;
    next = ptr;
  }
}
```

Build a list - with constructor

```
 string name;
 NameNode * names = NULL;
 NameNode * temp = 0;
 while (cin >> name)
 {
   temp = new NameNode;
   temp->name = name;
   temp->next = names;
   names = temp;
 }
```