CSE 333 – SECTION 5

C++ and Midterm Review

Overview

- C++ Classes, Constructors, new, delete, etc.
- Drawing Memory Diagrams
- Midterm Topic List

C++ classes

- Encapsulation and Abstraction
- Access specifiers:
  - Public: anything outside the class can access it
  - Protected: only this class and derived classes can access it
  - Private: only this class can access it
- Polymorphism
- Multiple Inheritance

new and delete

- new is used to allocate objects and primitive data types on the heap
- delete is used to deallocate these heap allocated objects
- Use “delete [ ] array” on an array
- Unlike malloc() and free(), new and delete are operators

Initialization vs Assignment

```
#define MAXSIZE 3

class IntArrayList {
public:
  IntArrayList() : array_(new int[MAXSIZE]), len_(0), maxsize_(MAXSIZE) { }

  IntArrayList(const int *const arr, size_t len) : len_(len), maxsize_(len_*2) {
    array_ = new int[maxsize_];
    memcopy(array_, arr, len * sizeof(int));
  }

  IntArrayList(const IntArrayList &rhs) {
    len_ = rhs.len_;
    maxsize_ = rhs.maxsize_;
    array_ = new int[maxsize_];
    memcopy(array_, rhs.array_, maxsize_ * sizeof(int));
  }
  ...
private:
  int *array_;
  size_t len_;
  size_t maxsize_;
};
```

Memory diagram

Memory diagram

```
class Wrap {
public:
  Wrap() : p_(nullptr) { }
  Wrap(IntArrayList *p)
    : p_(p) { *p_ = *p; }
  IntArrayList *p() const
    {return p_;}
private:
  IntArrayList *p_;
}

struct List {
  IntArrayList v;
}
```

```
int main() {
  Wrap a;
  Wrap b(new IntArrayList);
  struct List c { };
  struct List d {*b.p()};
  a = b;
  c = d;
  Wrap *e;
  e = &a;
  Wrap *f = new Wrap(&d.v);
  struct List *g =
      new struct List;
  g->v = *(new IntArrayList);
  delete f;
  delete g;
  return 0;
}
```

Operator Overloading

- A form of polymorphism.
- Give special meanings to operators in user-defined classes
- Special member functions in classes with a particular naming convention
- For E.g., for overloading the ‘+’ operator, define a member function named operator+

Common operators

- The most commonly overloaded operators are
  - = (assignment operator)
  - + - * (binary arithmetic operators)
  - += -= *= (compound assignment operators)
  - == != (comparison operators)

Midterm topic list

- **General program organization and where C fits in the ecosystem**
- System layers: C language, libraries, and operating system
- General workflow needed to build a program – preprocessor, compile, link
- Preprocessor – how #include, #define, #ifndef and other basic commands rewrite the program
- Structure of C/C++ programs: header files, source files
  - Declarations vs definitions
  - Organization and use of header files, including #ifndef guards
  - Faking modularity in C – headers, implementations
  - Internal vs external linkage; use of static for internal linkage
  - Dependencies – what needs to be recompiled when something changes (dependency graph behind make and similar tools)
  - Make and makefile basics – how build dependencies are encoded in makefile rules

Topic List (Contd.)

- **C language and program execution**
- Review: standard types, operators, functions, scope, parameters, strings, etc.
- Extended integer types (int32_t, uint64_t)
- Standard I/O library and streams: stdin, stdout, fopen, fread, scanf, printf, etc.
- POSIX libraries – wrappers for system calls
  - POSIX-layer I/O: open, read, write, etc.
  - Relationship between C standard library, POSIX library functions, and system calls
- Error handling - error codes and errno
- Process address space and memory map (code, static data, heap, stack)
  - Object lifetimes: static, automatic, dynamic (heap)
  - Stack and function calls – what happens during function call, return
- Function parameters
  - Call by value semantics (including structs, pointers)
  - Arrays as parameters - pointers
  - Using pointers for call-by-reference semantics
  - Function pointers as parameters

Topic List (Contd.)

- Pointers, pointers, pointers - &, *, and all that
  - Typing rules and pointer arithmetic (what does p+1 mean?)
  - Relationship between pointers and arrays, a[i] and pointer arithmetic
  - String constants, arrays of characters, C string library
  - Using void* as a “generic” pointer type
  - Casting
  - Dynamic allocation (malloc, free)
  - Potential bugs – memory leaks, dangling pointers (including returning pointers to local data), etc.
  - Be able to draw and read diagrams showing storage and pointers, and be able to trace code that manipulates these things.
- Structs – how to define and use, meaning of p->x (= (*p).x)
- Typedef – how to define and use
- Linked data structures in C – linked lists, hash tables, etc.

Topic List (Contd.)

- **C++**
- Classes and modularity, namespaces
  - Be able to read simple class definitions and add to them, implement functions, trace code, etc.
  - Know the difference between constructors, copy constructors, and assignment and when these are called
  - Know what a destructor is and when it gets called
- Other basic differences from C
  - Simpler, type-safe stream I/O (cout, cin, << and >>)
  - Type-safe memory management (new, delete, delete[])
  - References – particularly reference parameters
  - More pervasive use of const (const data and parameters, const member functions)