C++ aggregates

Plan

• Aggregate Data Structures
  – struct
  – union
  – arrays

struct

• Collection of variables organized into a data structure
• Predecessor to the class

Declaring a struct

```cpp
class Date {
    int month;
    int day;
    int year;
};
```

struct variables

```cpp
struct Date today;  // C style
Date yesterday;     // C++ style
```

Accessing struct members

```cpp
Date today;

today.month = 9;
today.day = 12;
today.year = 2005;
```
Initializing a struct

Date today = {9, 12, 2005};

Structures within structures

```c++
struct Date { int month, day, year; }
struct Employee {
    int number;
    double salary;
    Date date_hired;
};

Employee fred;
```

Structures within structures

```c++
Employee fred;
fred.number = 42;
fred.salary = 32000.0;
fred.date_hired.month = 9;
fred.date_hired.day = 9;
fred.date_hired.year = 2000;

Employee barney = {43, 34444, {9, 9, 2001}};
```

Structs and functions

```c++
// Note parameter is still pass by value
void scheduleParty (Date d) {
    cout << "The party is scheduled for " << d.month << ":" << d.day << ":" << d.year << endl;
}

Date today = {9, 12, 2005};
scheduleParty (today);
```

Structs and pointers

- You can declare pointers to struct variables.
- To access a member from a pointer use the member access operator (->)
Structs and pointers

Date today;
Date *goodday;
goodday = &today;
goodday->month = 9;
goodday->day = 12;
goodday->year = 2005;

Date today;
Date *goodday;
goodday = &today;
(*goodday).month = 9;
(*goodday).day = 12;
(*goodday).year = 2005;

// Note parameter is still pass by value
void getPartyDate (Date *d)
{
    d->month = 9;
    d->day = 12;
    d->year = 2005;
}

Date party;
getPartyDate (&party);
scheduleParty (party);

Structs and pointers

The union

• Like a struct, but…
  – Data members share memory
  – Can contain only one value at any one time
  – Size is size of largest member.

union Holder
{
    char holdchar;
    short int holdshort;
    int holdint;
    double holddouble;
};
The union

```c
struct Variable {
  int type;
  union Holder value;
};

void printVariable (Variable V) {
  switch (V.type) {
    case 0:  cout << "value " << V.value.holdchar << endl;
    case 1:  cout << "value " << V.value.holdshort << endl;
    case 2:  cout << "value " << V.value.holdint << endl;
    case 3:  cout << "value " << V.value.holddouble << endl;
    default: cout << "Invalid type" << endl;
  }
}
```

Arrays

- Aggregate collection of items in contiguous memory
- Array variables always pointers... – I.e. The contain the memory address of the first element of the array
- Array elements are accessed using the [ ] operator
- Array element indices begin at 0.

```c
Arrays

- Ways of declaring arrays
  - If you know the number of elements in the array
    - int myArray[7];
  - If you want to initialize the array when declared
    - int myArray[] = { 1, 2, 3, 4, 5, 6, 7 };
  - Dynamic allocation on heap
    - int myArray[] = new int[3*n];
```

```
Arrays

0x12332 0x123332
0x123332[0][1][2][3][4][5]
a
```

```
The union

Holder hld;
hld.holdchar = 'a';
hld.holdshort = 5;
hld.holdint = 12345;
hld.holddouble = 7.;

Note: programmer must keep track as to which union datatype is being used at any one time
```
Array variables

```c
short a[] = { 11, 2, 23, 42, 15};
short *b = a;
short *c = &a[0]);
```

Arrays

- Especially true when passing to function
```
int a[]= {1, 2, 3, 4};
void foo (int b[])
foo (a);
{
    b[2] = 12;
}
```

Arrays

- Works just as well
  - In fact, arrays are converted to * when given as arguments
```
int a[] = {1, 2, 3, 4};
void foo (int *b)
foo (a);
{
    b[2] = 12;
}
```

Pointer Arithmetic

```
short a[] = { 11, 2, 23, 42, 15};
short *d = &(a[2]);
short *e = a + 2;
short dval= a[2];
short eval = *(a +2);
```

Arrays

- Unlike Java, C++ arrays have no bounds checking.
```
int a[] = {1, 2, 3, 4};
void foo (int *b)
foo (a);
{
    b[7] = 12;
}
```

Arrays and Strings

- C-style string
  - Strings are represented as array of char terminated by a ‘\0’
  ```
  char *myName = "Joe";
  char myName[] = "Joe";
  ```
  ```
  • myName[0] = 'J'
  • myName[1] = 'o'
  • myName[2] = 'e'
  • myName[3] = '\0'
  ```
Arrays and Strings

- Can manipulate C style strings using strings library:
  - `char *strcat (char *s1, const char *s2)`
  - `int strcmp (const char *s1, const char *s2)`
  - `char *strcpy (char *s1, const char *s2)`
  - `char *strchr (const char *s, char c)`

- Complete list can be found by:
  - `man -s 3S string`

Arrays of struct

```c
struct Date { int month, day, year; }
Date c[5];
c[0].month = 9;...
```

Arrays of pointers to structs

```c
struct Date { int month, day, year; }
Date *c[4];
c[0]= new Date;
c[0]->month = 9;...
```

Arrays and Strings

- Another look at main()
  - `main (int argc, char *argv[])`
  - `foo 1 3 fred`
  - `argc = 4`
  - `argv[0] = “foo”`
  - `argv[1] = “1”`

Arrays and Strings

- Another look at main()
  - `main (int argc, char *argv[])`
  - Interpreted as an array of arrays.
  - Or as an array of pointers to 1st element of arrays

Multidimensional Arrays

- C++ does support multidimensional arrays:
  - Interpreted as an array of arrays.
  - Or as an array of pointers to 1st element of arrays

  - Example:
    - `int a[5][10];`
    - `a` is an array of 5 arrays of 10 integers.
Multidimensional Arrays

- int a[5][10];

Arrays

- Questions?