

**CSE 459.22**  
**Programming in C++**

Neelam Soundarajan  
Computer Sc. and Engineering  
Dreese Lab 579  
Tel: 292 1444

Please send corrections by e-mail to  
neelam@cse

1

**Some Terminology:**

*Procedural programming:* Focus is on the algorithms necessary to do the required computation. Data structures introduced as needed.

*Modular programming:* The program is partitioned so that each set of procedures and the data they manipulate is grouped into a separate *module*. A given module may be changed without affecting the correctness of the other modules.

3

**Who I am:**

Faculty member in the CSE Dept.  
I usually teach CSE 655, 755.

**Who you are (or should be):**

CIS/CSE major who has completed CSE 321.

**Others:**

Must have a reasonable background in programming/software. This is *not* a beginning course in programming.

**Course focus:**

Important technical details of C++ including *some* “low-level” details.

Focus will be on the *core* language, not details such as differences between various implementations etc.

2

**Terminology (contd.):**

*ADT-based programming:* Often, *multiple* instances of a particular kind of data are needed. So: create corresponding types. The rest of the program *uses* the types but doesn't know the internal details.

*Object-oriented programming:* Somewhat similar but often the “objects” correspond to entities in the actual (real-world) system. Plus, some important technical differences.

*In most languages (including C++)* both ADT-based and OOP are implemented using `classes`.

**Reading:** Chapter 2.

4

## Types (Ch. 4):

### Basic types:

bool, char, int, double  
(and slightly fancier versions)  
enum ("enumerated" type).

### Pointer types:

int\* etc.

### Array types: ...

### Structure of a declaration:

Consists of an optional "specifier", a base type, a declarator, optional initializer.

Specifier is things such as "extern" – gives us some non-type attribute.

Declarator is the name and optionally some declarator operator such as:

\*, \*const, &, [], () (pointer, constant pointer, reference, array, function).

### Declaring multiple names:

```
int x, y; // same as int x; int y;  
int* p, q; // NOT same as int* p; int* q;  
int v[10], *pv; // int v[10]; int* pv;  
typedef int* IP;  
IP p1, p2; // Okay
```

## Types and declarations(contd.):

Before a name is used it must be *declared*.

```
char ch;  
string s; // Not a "built-in" type  
int count = 1;  
const double pi = 3.14;  
enum Months { Jan, Feb, Mar, ...};  
int lab1Grades[40];  
extern int errorNo; // Declaration  
const char* name = "C++";  
struct Date {int d, m, y};  
int date(Date* p) {return p->d;}  
double sqrt(double); // Declaration  
typedef int MyInt;  
  
const c = 7; // Error! no type specified.  
gt(int a, int b) { return (a>b) ? a : b; }  
// error: return type?
```

5

6

## Scope:

```
int x; // global x;  
  
void f() {  
    int x; // local x; hides global x  
    x = 1; // assign to local x;  
    {  
        int x; // hides first local x;  
        x = 2; // assigns to this x;  
    }  
    x = 3; // assigns to first local x  
}
```

```
int* p = &x; // addr. of global x
```

**Exercise:** Read Section 4.9.5 (Initialization).

**Exercise:** Do problem (1), Section 4.11 (p. 85).

7

8

## Pointers, Arrays, Structures (Ch. 5):

For any type T, T\* is the "pointer to T type".

```
int* pi; // pointer to int
char** ppc; // pointer to pointer to char
int* ap[15]; // array of 15 pointers to ints
int (*fp)(char*); // ptr. to fn. taking a char*
arg;
int* f(char*); // fn. taking a char* arg., returning
ptr. to int

char c = 'a';
char* p = &c; // p holds addr. of c
char c2 = *p; // c2 == 'a'

void* vp; // can point to ANY type
```

## Pointers, Arrays, Structures (contd.):

For any type T, T[size] is the "array of size elements of type T", indexed from 0 to size-1.

```
double v[3]; // array of three doubles, v[0],
v[1], v[2]
char* a[32]; // array of 32 ptrs to char: a[0],
... a[31]

void f(int a) {
    int v1[i]; // illegal; size not a constant exp.
    vector<int> v2(i); // ok; uses STL
}

int d2[10][20]; // array of 10 arrays of 20 ints

int d3[10,20]; // error!
```

**Exercise:** Read Section 5.2.1 (initialization).