PROGRAM IN COMPUTING 10A
PROFESSOR POLLETT

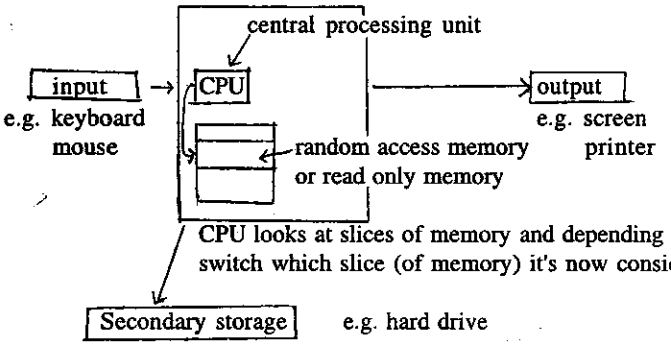
SPRING 2000

LectureNotes *Spring 2000*

PROGRAM IN COMPUTING 10A SECTION 3 & 4
PROFESSOR POLLETT
SET #1

APRIL 3, 2000

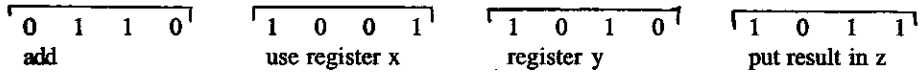
- Overview of computer



- common "slice" of memory
  - 8 bits which is 1 byte
  - a bit is a 0 or a 1.
  - words which are usually groups of 2 or 4 bytes

The CPU runs instructions in machine language
↑
written using strings of 0's and 1's

Example
instruction



(take what's in y, add it to what's in x, and put result in z)

The instruction ADD X Y Z. is an assembly language code for the above machine language 0's and 1's.
Writing in an assembly language is very tedious.

Would like to program in a language closer to English and translate result into machine code.
Example: C++ program which does translation is called a compiler

compilers take C++ programs and produce object code. If your code uses functions defined by somebody else, these won't be filled in by compiler.
A linker is used to fill-in references for other person's code.

• Simple C++ program.

```
#include <iostream.h>
void main ( )
{    cout << "hi there" << endl;
}
```

prints:

```
hi there
```

<iostream.h> header file: says what the functions somebody else has written should look like
iostream.h input/output header e.g. cout

void: means function main does not return a value
In a c++ program, main is always the first function that is run.
cout: prints "hi there".
to standard output (screen)
functions begin with { and end with }

END OF LECTURE **********

APRIL 5, 2000

Last Day: introduced our first c++ program
Today: give a slightly more complicated program
discuss commenting code
good code format
a little bit on the history of c++
differences from c
variable declarations

Let's look at a slightly more complicated program

```
// filename: area.cpp    ← "//" two slashes tells c++ compiler to ignore this line as far as compiling goes.
// purpose: takes two inputs from user in inches and computes their area.
// known bugs: none (if program does not work perfectly, comment on the known bugs/problems in your program)

// is a C++ comment
/* this is a comment */ ← C style comment
        ↑
        anything written here will be ignored by compiler
    (C style comment is useful for several lines of comments. ex: /* .....
                                                                   .....
                                                                   ..... */
                                        could cause error
    Note:   /*      /*      */      */
(nested comments)           (comment ends here)

#include <iostream.h>
int main ( )
↑
main will
return a
value to                usually, if a 0 is returned, tells the operating system that program worked OK.
the operating
system
(ex. windows,
UNIX ...)

int main ( )            { } → stuff between a pair of braces called a block
{
    int width, height, area; // declare 3 variables of type int
                                                    ↓
                                                    a string of at most 32 1's and 0's
    cout << "Enter a width in inches\n";  /* - all statements in C++ have to end in a semicolon
                                            - \n means carriage return
                                            we could have done << endl;
                                            - cout: means direct string to current output stream */
    cin >> width;
    cout << "Enter a height in inches\n";
    cin >> height;
    area = height * width;              //take value of height * width and stores it in the variable area.
    cout << "The area is " << area << " square inches";
                    ↑           ↑
```

```
    return 0;                          // return 0 → no error
}
```

Prints:

```
Enter a width
11
Enter a height
7
The area is 77 square inches
```

11 and 7 given/entered by the user

- little bit of history of C++

Algol — early 60's
↓
B
↓
BCPL } developed in tandem with Unix, Ritchie Thompson, early 70's
↓
C
↓
C++

simula — early object-oriented language, 60's, 70's
↓
small talk — (xerox parc) 70's, 80's

Stroustrup thought object oriented ideas were cool, and hacked them into C to make C++

- C versus C++
- in general, any C code is C++ code.
- C++ added ways to define complex data types called classes; also added operator overloading
- input/ouput based on Unix redirects new to C++

cin, cout → in C++ but not in C

In C, would use — integer

```
printf ("hi there %d", i);
                  ↑
          fill in integer
```

to print to screen

To read an input, would do

```
scanf ("%d", &i);
        ↑    ↑
```

type is an int stick input into i

- C++ slightly more flexible on where variable declarations can be done.

But for this class, will not use this.

for us, a simple C++ program will look like:

```
header includes
int main ( )
{
    variable declarations    // in C, variable declarations must be at start of block
    statement_1;             // in C++, can move variable declarations to different places.
    statement_2;
      ⋮
    return 0;
}
```

END OF LECTURE **********

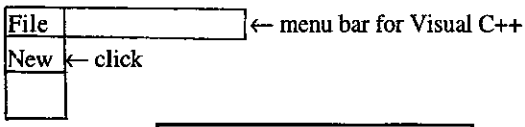
April 7th

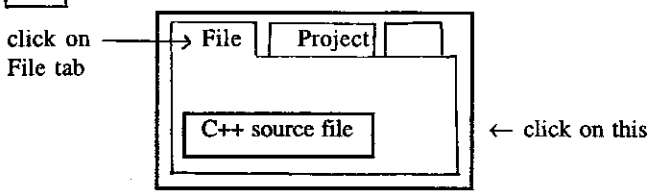
Last Day: Comparing C++ to C.
Today will very briefly describe how to use visual C++.

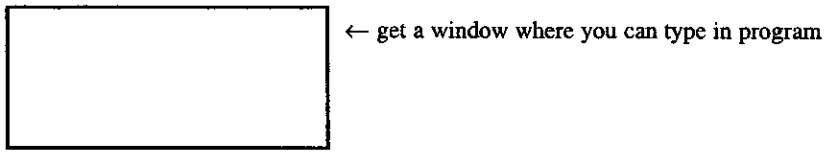
will also talk about variable declarations in C++.
variable assignments in C++.
input/output.

To run Visual C++:
Click on start → Visual Studio 6 → click on Visual C++

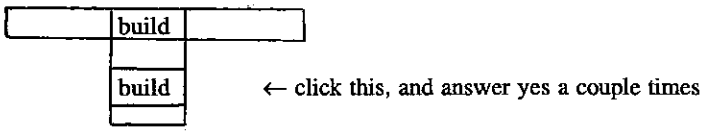
To create a new C++ source file



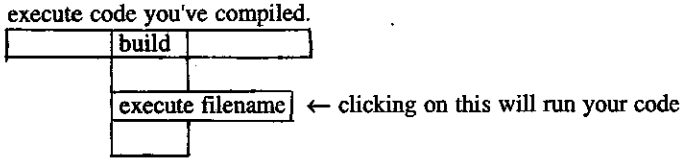




To save what you type, click on file, then click on save or save as.
To open an existing file, go to file, click on open.
To compile your program.



To execute code you've compiled.



Variable declarations:

```
    int  dontRemember;
type↑    variable name ↑
```

What constitutes a legal variable name?
identifier

-Can be upper or lower case
-After first letter or underscore, can use letters or numbers or underscores
example:

```
x_s
x100
hi_there
_ImAGlobal
```

↑ In old days of C, variables beginning with underscore were used for global variables
(a global variable is one that can be used in all functions).

Note: names Bob, boB, and BoB are all different (case matters).

Example of illegal variable names:

12	not variables!	cannot start with a number
3x		
%hi		cannot start with a %.
prog.c		cannot have .

Common conventions for variable names:
-i,j,k for counters
-otherwise, variable names should be a descriptive string of words
first word all lower case
first letter of subsequent words capitalized
example:
selfDestructFlag (a flag is a variable that can be true or false)
-For global variables, put a g at start.
example:
gImGlobal

Assignment Statements
example:
double tommy; (double=double precision float/decimal).
tommy = .995;
↑ expression in this case is a constant (a number).
lvalue of
assignment ("left hand value")
(variable that will get a value from right hand side).
This stores .995 in memory locations for a double named tommy
example:
int distance, dist2, vel=5, time(3); same as time = 3 (writing time(3) is much less common than time=3)
can do assignments in declarations.
distance=vel*time; //stores 15 in distance.
distance=distance+1; //stores 16 in distance.
dist2=dist2+1; //not valid at this point since dist2 has not been given a value yet.

Input/Output in C++
Sundry things we haven't mentioned yet . . .
cout<<"hi there\n"//\n: escape sequence for a new line.
<<"you"; //legal to put redirects over several lines.

Other escape sequences:
\t - tab
\\ -prints \
\" -prints a double quote
\a -alert(goes "ding") → computer makes an alert sound like "ding."
\z -z (other letters just don't print \)

END OF LECTURE AND SET #1********

PROGRAM IN COMPUTING 10A SECTION 3 & 4
PROFESSOR POLLETT
SET #2

APRIL 10, 2000

Last day: assignments/declarations, input and output again.
Today: finish up talking about input/output, data types, mixing types, arithmetic expresions, assignment abbreviations.

• How do you print out doubles to some fixed precision?
suppose the price is $5, and the tax is 8%

```
double price = 5, tax = 0.08;
cout << "Tax" << (price * tax) << "\n";
```

will get
Tax 0.4000
↑
too many digits

```
cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);
```
↑
number of decimal places precision.

```
cout << "Tax" << (price*tax) << "\n";
```

In this case would print 0.40

Data Types and Expressions

1) 2 is different from 2.0
2 is an int
2.0 is a double
2 is accurate to an infinite number of digits.
2.0 is accurate to the precision the computer uses to store doubles.
How are doubles stored?
Roughly, a double is stored as a mantissa and an exponent.
↑
string after a decimal
(0.21, -7)
↑
might represent 0.21×10^{-7} → like scientific notation
(precision is the number of digits mantissa can hold)

(Aside: to write things in scientific notation in C++ ...
2.1×10^{7} would be written as 2.1e7.)

Data Types in C++

	size	range	precision
short	2 bytes	-32767 to 32767	N/A
(short int)		(2^{16}: 2 bytes - each byte has 8 bits → 16 bits → each bit can be 0 or 1 → 2^{16})	
		↑ high order bit used for -	
int, long (long int)	4 bytes	-2,147,483,647 to 2,147,483,647	N/A

float	4 bytes 10^{-38} to 10^{38}	to 7 digits precision

↑
called because decimal point "floats"

double	8 bytes 10^{-308} to 10^{308}	to 15 digits
long double	10 bytes 10^{-4932} to 10^{4932}	to 19 digits

char 1 byte
↑
used to store ASCII characters
↑
standard from 60's to encode keyboard symbols. A = 65

```
char a = '+';   // notice for characters, use single quotes.
cout << a << "\n";
```
would print + and a carriage return to screen.

bool type
variable of this type can be either true or false.
e.g.
```
bool flag1=true, flag2=false;
```

Mixing Types
generally, can use "less complicated" types in right hand side of assignments with lvalue a "more complicated" type.

e.g.
```
int i=1, y;
double m=21.1, x;
x = i + m;          // is okay
y = i + m;          // not okay as y "less complicated" than m.
```

Arithmetic expressions

functions built out of +, -, *, /, %, variables, and constants.
↑
returns remainder
e.g. 25%5 returns 0
5%4 returns 1.

e.g.
```
5 + x*x
```

*, /, %, +, -
highest precedence lowest precedence

so 5 + x*x = 5 + (x*x)
operators associate from left to right.
5 - 3 - 2 = 0
(5 - 3) - 2 = 0
not 5 - (3 - 2) = 4

use parantheses when you want to force the issue.
e.g. (x + y + z) / 3
would give the average of x, y, z.

More on assignments
rather than write
```
count = count + 2;
```
can abbreviate this as
```
count += 2;
```

similarly, there are %=, *=, /=, -= assignments.

e.g.

salary * = 1.1;
total = total - discount;
same as
total -= discount;

END OF LECTURE **********

APRIL 12, 2000

Last day finished by talking about +=, *=, -=, %=, /=.

e.g.

total = total - discount;
total -= discount;

Today will talk about conditionals in C++.

conditionals - a way to get the computer to choose between more than one alternative.

e.g.

Suppose you wanted to calculate tax based on income.
People making < 40,000 taxed at 20%.
If making ≥ 40,000, taxed at 30%.

In C++ could do

```
if (income < 40000)
    tax = income * .2;
else tax = income * .3;
```

Semantics of "if":

↑

meaning

```
if (condition)          // if boolean condition is true, do statement 1.
    statement1;
else                    // otherwise do statement 2.
    statement 2;
```

income < 40000 is a boolean condition.

Other conditionals:

```
==  ,   !=  ,   <=  ,   >=
 ↑       ↑
```

equals not equals

Can make more complicated conditions out of simple ones.

e.g.

```
# include < iostream.h>
int main ( )
{
    int day, favNumber, garbage;
    cout << "Please enter day of Month: ";
    cin >> day;
    cout << "\n\n Please enter favorite number:";
    cin >> favNumber;
        ↓ causes less errors to write 17 on left- hand side
    if  (17 = = day &&  7 == favNumber || day > = 25)
                                          ↑
                                          or
{
    cout << "\n the planets are aligned.  Type something \n";
    cin >> garbage;
}
    return 0;
}       // end code.
```

Prints:

Please enter day of month: 17

Please enter favorite number:
The planets are aligned. Type something.]-would not be printed if favNumber was 6, and day was 17.
83842]-would be printed if day was 25 and favNumber was 6

type bool

x	y	x&&y
true	true	true
true	false	false
false	true	false
false	false	false

x	y	x||y
true	true	true
true	false	true
false	true	true
false	false	false

Negation!

x	!x
true	false
false	true

```
bool flag1 = true, flag2 = false;
if (!(flag && flag2))
    {cout <<"hi there";
    }
```
would print
hi there

- Notice in very first example, had just a statement after if condition. Later used { } (compound statements or blocks) with a bunch of statements in them. Either way is good C++.
- Notice that you don't have to have an else clause.

More on &&, ||
eg.
```
    if (true || 7/0 > 1)          // 7/0 would cause an error if it had to be evaluated.
        cout <<"hi there\n";
```
would print   // But it is never evaluated since the condition is already true by the first condition.
hi there

So if there are a bunch of ||s
    cond1 || and cond2 || ... || condn
evaluate until you find first condition that is true and retrun true. If you don't find any true, return false.

For &&, search for first false and return false.
eg.
```
    if (false && 7/0 != 1)   // stop evaluating after false
        cout << "bogus";
    else if (true)
        cout << "unbogus";
```
would print
unbogus

Note: you can use an if immediately after an else.
    In general, can do
```
        if (cond1)
        {
            // stuff1
        }
        else if (cond2)
        {
            // stuff2
        }
```

```
        ↓
    else if (condn)
    {
        // stuffn
    }
    else
    {
        // whatever
    }
```

Consider:

```
(1) if (7 ==x)
        cout << "hi1";
    if (6==y)
        cout <<"hi2";
```

versus

```
(2) if (7==x)
        cout << "hi1";
    else if (6==y)
        cout << "hi2";
```

when x = 7, y = 6,
(1) would print
hi1hi2
(2) would print
hi1

END OF LECTURE **********

APRIL 14, 2000

Last Day: Talked about conditionals and if statements in C++

```
if ( cond)
{
    // stuff
}
else
{
    // other stuff
}
```

Today: talk about two different looping mechanisms in C ++ while loops, do/while loops. Then talk about increment and decrement operators. Then infinite loops.

Looping Mechanism- is a program structure used to execute a block of code over and over till some condition is met

Example:

```
// Program Name : power.cpp
// Purpose: Takes a number n from user and computes 2^n
// Known bugs: works only for whole numbers.

# include  <iostream.h>
int main ( )
{
    int number, power  = 1, counter = 0;
    cout << "Enter a number: " ;
    cin >> number;
    while ( counter < number)
    {
        power * = 2;
        counter +  = 1;
    }  // check condtion again.

cout << "Power is:" << power;
```

```
return 0;
} // end of main
```

Enter a number: 3 ← entered by user

not printed:
- number = 3, power = 1, counter = 0
- the while condtion is satisfied. (0 < 3)
- number = 3, poewr = 2, counter = 1
- the while condition is true (1 < 3)
- number = 3, power = 4, counter = 2
- the while condition is true (2 < 3)
- number = 3, power = 8, counter = 3.
- the while condition is false

Power is : 8
(program ends)

Number of loops needed to computer 2^n is n.
we're using $2^n = 2(2^{n-1})$
Could have written a faster program if used

$2^n = (2^{n/2})^2$ if n is even

$= 2 \cdot (2^{n/2})^2$ is n is odd

$\lfloor \frac{N}{2} \rfloor$ $\frac{N}{2}$ rounded down

could use this to write a program that only had to loop log many times. (talk more about later).

• General Form of a while loop

```
while ( cond)
    statement;
```
OR
```
while (cond)
{                   executes stuff over and over until cond. becomes untrue.
    // stuff
}
```

- using what we've learned so far of C ++:
assignments, if , while-loops
we could in theory write a program equivalent to any program in all of C++ (ignoring input/output).
- The point of adding more stuff to the language is to make it easier to understand programs.

• do/while loops
like a while loop except guaranteed to execute the statements in its block at least once.

Example:

```
// HiThere.cpp
# include <iostream.h>
int main ( )
{
    char c ;
do
{

cout << "Hi There. If you want to see this greeting again type y \ n";

    cin >> c;
```

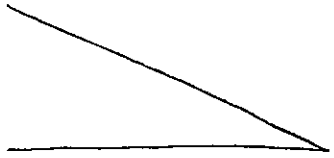
```
    } while ( c == 'y');  // need semicolon

    return 0;
    } // end of main
```

Prints:

```
Hi there. If you want to see this greeting again type y
y
Hi there. If you want to see this greeting again type y
n
program ends
```

• General Form of do while loop

```
    do
        statement;
    while (cond);
OR
    do
    { // stuff
    }
    while (cond);
```



semicolon needed.

semantics: execute statement at least once. Then keep executing statement again and again until cond becomes false.

Increment/Decrement operators

- In first while program today.
  had line counter + = 1;

  will be translated to an ADD
  machine language usually has a faster statement for adding one called INCR
  To hint to compiler to use this and also to save typing, can do:

```
        counter++;    // means counter = counter +1;
```

similarly for subtracting by one, can do:

```
        counter -- ;  // means counter = counter -1;
```

Infinite loops

```
while (true)
{
    // stuff
}
```

keeps executing stuff over and over forever
(sometimes useful)

whats bad is if you have a cond that always evaluates to true but didn't intend it.

END OF LECTURE AND SET #2 **********

LectureNotes

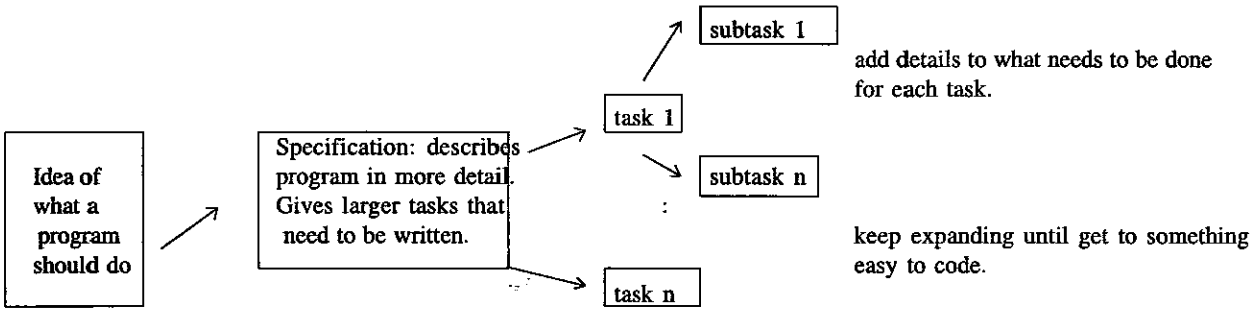
Spring 2000

PROGRAM IN COMPUTING 10A, S.3,4
PROFESSOR POLLETT
SET #3

APRIL 17, 2000

Last day - recall we're talking about while loops/do while loops.
said fragment of C++ using only assignment, if, while was computationally as powerful as all of C++.
Today - we look at one mechanism which although it does not increase the computational power of C++, will allow us to write large understandable programs, i.e. will talk about functions.

A common methodology for writing larger programs is known as stepwise refinement.
In it....



Different groups can work on writing the different tasks. If specification is detailed enough, other groups can write their code assuming the other groups' code is usable in theirs.

Functions will allow us to break code up into these tasks from stepwise refinement model.

General Idea of a function

```
TelephoneNumber (fName, lName)    //Given a first name and a last name
                                  //Checks some files on disk and outputs a
Naming convention for             //telephone number
functions: upper case first letter of each word.
```

Other groups don't particularly need to know how this program works in order to use it.

Basic components of a function

Identifier - its name
underscore or letter followed by letters, numbers, or underscores
Inputs - zero or more
For example above, fName and lName are inputs (arguments)
Output - zero or one value
returned by the function
For example above, telephone number was the output.
Side effects - function might change some memory on disk or in RAM. Most common side effect is print something to screen.

Example:
main( ) - has no inputs. We've had the output of main be both an int and void - (void case means no output). Have done printing in main se have seen side effects.

Example:
How to use someone else's function.
Using math functions from math.h

```
// Program Name: surveyor_cpp
// Purpose: Given distance to summit and an angle, computes height of mountain
// Known bugs: none

#include <iostream.h>
#include <math.h>      //tells compiler to look for math library functions.
int main( )
{
    double distance, angle;
    cout << "Enter distance to mountaintop: ";
    cin >> distance;
    cout << "Enter an angle: ";              //in radians
    cin >> angle;
    cout << "The height of the mountain is "
         << sin(angle)*distance;
        //Notice format to use a function.
    return 0;
}
```

Prints:
Enter distance to mountaintop:1
Enter an angle: .5
The height of the mountain is .5

Other math.h functions:

```
double sqrt(double x);        }whole thing called a function prototype
   ↑                          //computes square root of x
type of output

double pow (double x, double y);      //computes x^y

   sin
   cos
   tan

int  ceil (double x);         //rounds up to next bigger int, eg. 4.3 → 5
int  floor (double x);        //rounds down to next lower int, e.g. 4.3 → 4
```

fabs - computes absolute value.

Suppose wanted to write own code for absolute value.

```
#include<iostream.h>
int abs(int x);          //prototype tells compiler type of function, so can use
                         //it before it's defined.
```

END OF LECTURE **********

APRIL 19, 2000

Last day: started talking about functions
gave example of using other people's functions, i.e., math library.
Today: will write our own functions
talk about local variables

Code for absolute function

```
# include <iostream.h>
int abs (int x);     //function prototype
```

```
                    //says types of function but no code.
int main ()
{
    int number;
    cout <<"Enter an integer to test abs function:";
    cin >> number;
    cout << "The absolute value of " <<number <<" is " <<abs (number);
    return 0;
}

int abs (int x)
{
    if (x<0) return -x;        //value returned by function
    return x;
}
```

Prints:
Enter an integer to test abs function: -8
The absolute value of -8 is 8

What happens?

The value of number is passed to abs. A new variable x is created and set to -8. The if condition holds, so 8 is returned.

Another example of using someone else's function.
Consider:

```
    # include <iostream.h>
    int main()
{
    cout <<9/2 <<endl;
    cout <<double (9)/2;          //double is a built-in function
    return 0;                     //converts its input to a double.
}
```

Outputs:
4 (9/2 : both are ints and will output an int)
4.5

double function used to change the type of an integer to a double. This kind of change of type is called type casting.

Notice in double (9)/2, compiler will also convert the type of 2 to a double. This implicit change of type is called type conversion.

Another example of writing your own function

```
//Program Name : compounder
//Purpose : Takes an interest rate, a number of years, and an amount,
//          and outputs total you would get compounding the interest
//          that many years.
//Known bugs: 15% should be entered as .15

#include <iostream.h>
double Compounder (double i, int y, double a);
// calculates amount after a is compounded by rate i for y years.

int main ()
{
    double interest, amount;
    int years;
    cout <<"Enter interest rate:";
    cin>>interest;
    cout<<"\nEnter years:";
    cin>>years;
    cout<<"\nEnter amount:";
    cin>>amount;
    cout<<"The amount will be "<<Compounder(interest, years, amount);
    cout<<"It was originally "<<amount;
    return 0;
```

```
}
double Compounder (double intrst, int yrs, double amt)
{                                          ↑
    while (yrs > 0)                        local variable
    {
        amt * = (1+intrst);
        yrs --;
    }
    return amt;
}
```

Output:
Enter interest rate: .1
Enter years: 2
Enter amount: 1000
The amount will be 1210

What happens?
calls the function Compounder. The values of interest, years, and amount
(not the variables themselves) passed to function. This is called call by value. The variables intrst, yrs, amt are created, then assigned intrst = .1, yrs = 2, amt = 1000.
First time through loop:
amt = 1100
yrs = 1
Second time through loop:
amt = 1210
yrs = 0
end loop, return 1210.

It was originally 1000

- Notice amount printed out was 1000, unchanged by the call to Compounder.
- Order of arguments matter.
  i.e. Compounder (years, interest, amount),
  would get an error since interest not type int

END OF LECTURE********

APRIL 21, 2000

Announcements:
Practice Midterm is on web.
One problem from this is on actual midterm

Last Day: was talking about functions.
mentioned local/global variable
Today: going to talk about local/global variables
talk about const type
Function overloading

Example of local/global variables

```
#include <iostream.h>
int Square (int n);          // this function returns n * n.
void PrintOut (int, int);    // takes two integers, prints them out,
                             // returns nothing.
                             // Notice allowed to omit variable names
                             // in prototype.
int number;                  // create a variable number. Notice this variable does
                             // not appear in { }. So it's global.
int main ( )
{
    int num2 = 2;            // local variable of main
                             // declared in { } so it's local.
```

```
        number = 1;         // no number in main, so look in surrounding and
        {                   // see that number is the global variable.
            int number = 4;     // create a local variable to this block.
            PrintOut (number, num2);        // look for number in this block, = 4
        }   // end block
        PrintOut (number, num2);
        number += 2;            // so now number = 3.
        number = Square (number);       //now number = 9.
        PrintOut (number, num2);
    }   //end main

void PrintOut (int n1, int n2)              //the scope of n1 and n2 is the
{                                           //braces following.  They are local
                                            //variables.
    cout << "number 1:"<<n1 <<"\n number 2:"<< n2;
}

int Square (int num2)               //num2 is a local variable different from the
{                                   // one in main.
    int i = num2 * num2;
    return i;
}
```

Output:

```
number 1: 4
number 2: 2
number 1: 1
number 2: 2
number 1: 9
number 2: 2
```

Summary:

Every pair of { } and ( ) has associated with it a table of variable names and values.
When a variable is used, try to find its name in nearest enclosing table and either read or change value there.
Whole file also has a table. Variables in this table called global.
Scope of a variable is { } in which it was defined.

Type const

const variables' values can be read but not changed

e.g.

```
    #include <iostream.h>
    const double PI = 3.1415;               //global const
modifier                                    //value cannot be changed
⇒ modifies the type
    int main ()
    {   PI = 5;             //illegal.  cannot change the value of a const
        cout << PI;         //legal to print out.
        return 0;
    }
```

Function name overloading

- using the same name for more than one function
- Polymorphism (same sort of thing)
    - have a function which can operate on more than one type.

e.g.

```
    #include <iostream.h>
    const double PI = 3.1415;
    int Perimeter (int l, int w)
    {
        return (2*l + 2*w);
    }
```

```
    double Perimeter (double diameter)
    {
        return PI*diameter;
    }
//Notice no prototypes.  As long as you define the complete function before
//you use it, it's legal, but not too common because not as readable.

    int main ( )
    {
        cout << Perimeter (1, 3) <<"\n";
        cout << Perimeter (2.1) << "\n";
    }
```

Output
8
6.594

END OF LECTURE AND SET #3 **********

PROGRAM IN COMPUTING 10A S. 3,4
PROFESSOR POLLETT
SET #4

APRIL 24, 2000

Last day: finished by talking about overloaded functions
Today: will talk about overloading and type conversion, give an example of writing a program by stepwise refinement. Talk about call-by-reference.

Type conversion and Overloaded functions
e.g.

```
const double PI = 3,14;
int        Perimeter(int 1, int w)
     { return  2*l + 2*w;
     }
double Perimeter (double diameter)
     { return PI*diameter;
     }
```

What would happen if you call Perimeter (2)?
Will call double Perimeter (double diameter) function.

Basic idea:
If compiler can't find a function prototype exactly matching the input, it will try to apply type conversions to see if it can find a match.

Some more on stepwise refinement and void functions
Consider task of getting an input from user and if it's positive, print that many * and new line, and get another number.
Otherwise stop.

```
Enter a number
3
***
Enter a number
2
**
Enter a number
0
```

First pass using stepwise refinement.

```
# include <iostream.h>
// prototype
int main ( )
{ do { // get input from user
       // plot it
     } while (input > 0);
return 0;
}
```

Second pass:

```
# include <iostream.h>
int  GetInput ( );
void Plot(int column);
int main ( )
{    int input;
```

```
        do  { input  =  GetInput ();          notice plot is always called after GetInput
              Plot(input);
            } while(input>0);
    return 0;
    }
```

First pass GetInput.

```
int GetInput ( )
    { // request input
      // get it
      // return it
    }
```

Second pass GetInput

```
int GetInput ( )
    {   int in;
        cout << "Enter a number\n";
        cin  >> in;
        return in;
    }
```

First pass Plot

```
void Plot (int col)
{   // if col> 0, loop printing *'s and decrementing col.
    // print new line
}
```

Second pass plot

```
void plot (int col)
{   while (col>0)
    {   cout << "*";
        col - - ;
    }
    cout <<"\n";
    return;         //notice can use a return with no argument in a void function.
}
```

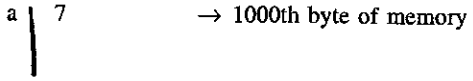
(1)shows how to break complicated tasks into simpler ones
(2)write functions to do simpler tasks

Although in most cases want only one exit point for our functions (since this is clearer), sometimes useful to exit in two places.

```
void function (// types)
{

    while(cond)  {   if (x = = 7) return;
                 }
    return;
}
```

At this point, all our functions return at most one value. How can we write functions to return more than one value? Use call-by-reference.

Rather than calling a function with the value of a variable, instead, call the function with the memory location of where that value is stored. (call-by-reference)



e.g.

```
    #include <iostream.h>
    void Triple   (int & num)        //&: use call-by-reference
    {
        num*=3;
    }
    int main ( ) {
```

```
    int x = 5;
    cout << "x = " << x << "\n";
    Triple(x);
    cout << "x=" << x << "\n";
    return 0;
}
```

Output:

x = 5 — If x stored at memory location 1000, then num will also point to location 1000.

x = 15 — So write 3*5 to memory location 1000.

END OF LECTURE **********

APRIL 26, 2000

Midterm Friday
Bring Photo ID
No cell phones
Closed book, closed notes
50 minutes
5 problems (covers up to 4.1) - one problem off practice midterm

Study strategy
1. Practice Midterm
2. Review HW1 and HW2
3. Look at your notes
4. Look at book

Practice midterm:

1) Define a bunch of terms with example.
   a) local variable - variable declared within { } or ().
      The value only exists as long as its block hasn't run to completion.

   e.g.

```
void MyPrinter (int i)
{
    double j;          //i and j are local variables
}
```

   b) type-casting - a programmer forced conversion of a variable from one type to another.

   e.g.

```
int x = 7;
double y = double (x);
             ↑
        typecast
```

   c) infinite loop - a loop that never terminates because its loop condition is always satisfied.
   e.g.

```
while (true)
{
}
```

   d) polymorphism - a function which operates on more than one set of input/output types.
   e.g.

```
int Perimeter (int l, int w)
{
    return 2*l + 2*w;
}
double Perimeter (double diameter)
{
    return 3.14*diameter;
```

```
}
```

2) Fragment 1

```
if (x == 1)
{
    cout<< "hi there";
}
else if (y==1)
{
    cout<< "ho there";
}
```

Fragment 2

```
if (x==1)
{
    cout<< "hi there";
}
if (y==1)
{
    cout<< "ho there";
}
```

Do they have the same output?
No, if x == 1 and y == 1, then Fragment 1 prints hi there
but Fragment 2 prints hi thereho there

3) int main ()

```
{
    int x = 0;
    if (false && (7/x > 0 || true))
        cout<< "hi there\n";
    return 0;
}
```

What does this print?
Prints nothing.

4)

```
#include <math.h>
double power (int x, int y)
{
    double out = 1, y2 = fabs (y);
    if (x==0) return 0.0;
    while (y2 > 0)
    {   out *= x;
        y2 --;
    }
if (y < 0) out = 1/out;
return out;
}
```

$2^{-3} \rightarrow 2^{3} \rightarrow \frac{1}{2^{3}} \rightarrow$ return this value

5)

```
#include <iostream.h>
const double salesTax = .0825;
int main ()
{
    double price;
    cout<< "Enter a price:";
    cin>> price;
    cout.setf (ios::fixed);
    cout.setf (ios::showpoint);
    cout.precision (2);
    cout<< "Your tax is: "
        << salesTax*price << endl;
    return 0;
```

```
}
```

6) Want to print out
This is a quote " and this is a slash \
To do this

```
#include <iostream.h>
int main ()
{
    cout<< "This is a quote"
    << "\" and this is a slash"
    << "\\ \n";
    return 0;
}
```

END OF LECTURE AND SET #4 **********

LectureNotes

Spring 2000

PROGRAM IN COMPUTING 10A
PROFESSOR POLLETT
SET #5

MAY 1, 2000

Last day - finished by talking about call-by-reference.
Today - more on call-by-reference, talk about pre-, post-conditions, and functions calling functions.

Call by reference

e.g. function to swap two numbers.
To test our function will write a short main program that creates two numbers, prints them, swaps them, prints again. Such short mains used to test a function are called a driver.

```
#include <iostream.h>
void swap(int & var 1, int & var2);
```

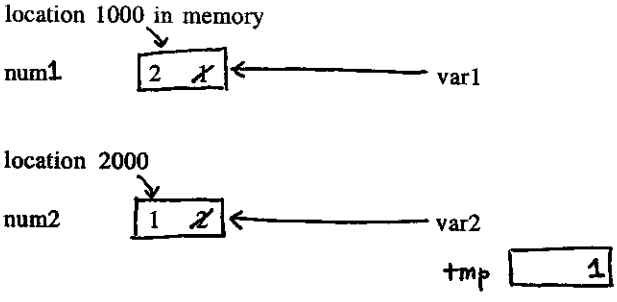
means call by reference
not legal in C

```
int main ( )
{
    int num1 = 1, num2 = 2;
    cout << "Test swap\n";
    cout << "\nnum1: " << num1 << "\n num2: " << num2 << endl;
    swap (num1, num2);
    cout << "\nAfter swap\n num1: " << num1 << "\nnum2: " << num2 << endl;
    return 0;
}
void swap (int & var1, int & var2)
{
    int tmp = var1;
    var1 = var2;
    var2 = tmp;
}
```

output:
Test swap
num1: 1
num2: 2
After swap
num1: 2
num2: 1

What happens?

num1 and num2 outside of swap function are affected by changes to var1, var2



call by value does not affect variables used to call a function
call by reference does affect variables used to call a function.

Besides driver programs, another useful tool in building large programs is a stub.
e.g. void swap (int & fnum, int & snum)
{ } // no code
A stub essentially has name of program and no body

Useful technique (for debugging): commenting prototypes.
Use pre-, post-conditions.

e.g. void swap (int & fnum, int & snum);
// precondition: fnum and snum have been given values.
// postcondition: values of fnum and snum have been interchanged.

Precondition - what's supposed to be true before a block of code executes.
Postcondition - what's supposed to be true after a block of code executes.

Example: call-by-reference, pre and postconditions, functions calling functions.
Problem: multiply 2 rational numbers and express in lowest terms.

↑
Q: fraction is an integer divided by an integer
e.g. $\frac{1}{2}, \frac{1}{3}$, ...

e.g. $\frac{2}{5} \cdot \frac{5}{3} = \frac{10}{15} = \frac{2}{3}$
↑
divide top and bottom by GCD (greatest common divisor)

```
#include <iostream.h>
void FracMult (int num1, int den1, int num2, int den2, int& resnum, int& resden);
//precondition: num1, den1, num2, den2 all have values and den1 and den2 are not zero.
//postcondition: resnum/resden is the result of num1/den1 * num2/den2 in lowest terms

int Gcd (int num1, int num2);

int main ( )
{
    int num1 = 1, den1 = 2;
    int num2 = 2, den1 = 3;
    int resnum, resden;
    cout << "Multiplying" << num1 << "/" << den1 << "and" << num2 << "/" << den2 << "yields";
    FracMult (num1, den1, num2, den2, resnum, resden);
    cout << resnum << "/" << resden << endl;
    return 0;
}
```

END OF LECTURE **********

MAY 3, 2000

Last day- did a longer example about multiplying 2 fractions and expressing in lowest terms.

Today- finish this example, talk about file I/O

What we've written so far.
A driver main used to test function FracMult.

It set up values num1 = 1, den1 = 2
num2=2, den2 = 3
created resden, resnum.
then it called FracMult (num1,den1,num2,den2,resnum,resden);
called by value call by reference

```
void FracMult ( int num1, int den1, int num2, int den2, int & resnum, int& resden)
{
    int gcd;
    resnum = num1 * num2;
    resden= den1 * den2;
    gcd = Gcd( resnum, resden); // calling a function from another
    resnum/ = gcd;
    resden/ = gcd;
}
```

Algorithm for GCD based on Euclid's Algorithm
uses: Gcd (a, b) = Gcd(b, a % b)

```
int Gcd ( int num1, int num 2)
{
    int tmp;
    while ( num2! = 0)
    {
        tmp = num1% num 2;
        num1 = num2;
        num2 = tmp;
    }
    return num1;
}
```

e.g. Gcd(10, 15)

First pass:	Second pass:	Third Pass:
tmp = 10	tmp = 5	tmp = 0
num1 = 15	num1 = 10	num1=5
num2 = 10	num2=5	num2= 0

So 5 is returned.

Output:
Multiplying 1/2 and 2/3 yields 1/3

Streams basic files I/0 (Chapter 5)

Will use to introduce idea of classes and objects.

A stream is a flow of characters or data.
If the flow goes into your program, it is called an input stream.
e.g. keyboard input, writing to a file.
If the flow goes out of your program, it is called an output stream
e.g. writing to screen, to a disk, or to a tape.
Useful to write/read to/from a file to be able to store permanently
high score lists, databases, etc.

How to do file I/O in C ++
e.g.

```
#include < iostream.h>
#include <fstream.h>  // used for files
int main ( )
{
ifstream   inStream ;
   ↑          ↑
a class      an object of type ifstream
```

```
ofstream  outStream;
int in, out;
cout  << " Enter an integer \n" ;
cin >> in;
outStream.open ( "file.dat");
                ↑
          calls the open function of the outstream object
outStream << in;   // writes to file.
outStream.close ( );      // closes file so others can use it.
inStream.open ( "file.dat");
inStream >> out;   // get an integer
inStream.close ( );
cout << " The integer was:"   << out << "\n";
return 0;
}
```

Output:
Enter an integer
5
The integer was:5

What happens?
Opens file.dat
writes 5, closes file.dat
open file.dat
read one int from start of file.
close file.

When a file is opened for reading, always start to read from beginning of file.

A class is a data type built out of smaller datatypes, and functions.
e.g. ifstream,ofstream

An object is an instance of a class.
e.g. inStream is an object of type ifstream.

A method is a function in a class.
To call a method, use dot operator.
e.g. outStream.open ("file.dat")
                ↑
          method of object

```
outStream.precision(2);
              ↑
          this would only affect outStream instance of ofstream.
```

END OF LECTURE **********

MAY 5, 2000

Last day- started talking about input/output stream classes.
Today- go more into detail on reading/writing files. Checking if file opened correctly, getting filename from user, what the magic formula (for precision, etc.) means.

How to check if a file opened correctly?
e.g.

```
ifstream inStream;
inStream.open ("file.dat");
if(inStream.fail())  // checks if file opened correctly
{

cout << " Could not open file\n";
exit (1);
}
```

exit () is in # include <stdlib.h>
exit returns control from your program to the operating system, no matter where it is called from.
The 1 means there was an error.

How to get user to choose filename.
e.g.

```
#include <fstream.h>
#include <iostream.h>
#include <stdlib.h>     // for exit ( )

int main ()
{  int in ;
 char  inFileName [16];   //  stores 16 characters, called an array
ifstream  inStream;
cout << "Enter a file name ( max 15 chars) \n";
cin >> inFileName;
// inFileName string automatically has a \0 appended to it so
// computer knows where string ends.
inStream.open (inFileName);
if(instream.fail())
{
     cout << " Could not open:"  << inFileName << endl;
     exit ( 1);
}
inStream >> in;
cout << in << endl ;
return 0;
}
```

If file1.dat has 7 in it, then
Output:
Enter a file name (max 15 chars)
file1.dat
7
(If file1.dat failed to open, would print "Could not open file1.dat")

Magic Formula can be used with ofstreams.
ie.

```
ofstream outStream;
 outStream.open ("hithere.txt");
outStream.setf (ios : : fixed);
 outStream.setf (ios : : showpoint);
outStream.precision ( 2);
outStream << 7.0;
```

This writes number 7.00 to file hithere.txt.

setf: stands for set flag. A flag is like a bool, can either be on/off.
(true/false)

ios : : fixed
input output class → fixed has some value defined in this class.
(value given in iostream.h).
ios : : fixed says doubles should be output in fixed point rather than scientific notation.
e.g. 20 rather than 2e1.

ios : : showpoint- forces decimal point to be printed for doubles.
e.g. 2.0 rather than 2

Other Flags

ios : : showpos - forces + or - to be printed.
e.g. + 7 rather than 7

ios : : scientific - forces scientific notation to be used
e.g. 2e0 rather than 2.

ios :: left - forces number fields to be left justified.
e.g. 7000␣␣␣␣ ← if field has width 8, number appears to left.

ios : : right- forces number fields to be right justified.
e.g.␣␣␣␣7000

These last flags are useful when printing forms.

```
Sales
   70
  100
100000
  500
```
← nice to have it all lined up

To set field width for one number
```
cout << "Seven";
cout.width (4);
cout << 7 << endl;
```

Print:
Seven␣␣␣7

Have to set width again to 4 for next number if you don't want to go back to default

END OF LECTURE AND SET #5 **********

PROGRAM IN COMPUTING, S. 3,4
PROFESSOR POLLET
SET #6

MAY 8, 2000

Last day: talked about files, streams, stream methods, and flags.
Today: will talk about manipulators, get, put methods, how to append to a file, end-of-file checking.

We showed we could control width of a field using the width method of an ostream. (parent of ofstream).

e.g. cout.width(4);
```
cout << "hi" << 7 << endl;
```

Prints:
hi ___ 7

A manipulator is a function we can directly feed into a stream.

```
cout << "hi" << setw(4) << 7 << endl;
                  ↑
```

manipulator: has same effect as cout.width(4);

Another example of a manipulator. Rather than cout.precision(4);, could do cout << setprecision(4);
Manipulators need #include <iomanip.h>

How to work on single characters?
For instance, using >> with cin or an ifstream, we miss special characters like end of lines.
Ans: Use get and put methods.

e.g. char c;
```
cin.get(c); //treats special characters like any other.
```

e.g.
```
cout << "Enter a line to be echoed: \n"
char sym;
do
{
    cin.get(sym);
    cout << sym;
} while (sym! = '\n');
cout << "done\n";
```

Output:
hithere
hithere - output like this rather than like hhiitthheerree
done

Two reasons for this:
input may be buffered.
output may be buffered.
A buffer is a place where data is stored while waiting to be processed.

To output a single character, use put.

e.g.
```
char a = ' ';
cout.put(a); // write a space to screen.
```

Can use get and put with ifstreams and ofstreams.

e.g.

```
ifstream in("file");
```

another way to open file. This is an example of using a constructor. Like the two lines ifstream in; in.open ("file");

```
char a;
in.get(a);
cout << a << endl;
// gets a character from file and prints to screen.
```

Sometimes useful to avoid advancing where we are in file.

If do
in.putback(a); //doesn't matter what a is.
//last character read in "putback" so that it will also be next char read.

Can think of putback as allowing you to back up one step in reading file.

More on writing to files:
So far when we write to files, we erase what was there.
If do

```
ofstream out;
out.open ("file", ios::app);
```

append flag.
then writing will be done continuing at the end of the file.

Another useful thing to be able to do with files we are reading is to be able to tell when we've reached an end-of-file.
Can use eof( ) method to do this.

eg. Program to count number of lines in a file.

```
#include <iostream.h>
#include <fstream.h>
bool NewLine (ifstream& in);
```

class: all classes must be passed by reference.

```
int main( )
{
    ifstream inFile;
    int count = 0
    char filename [16];
    cin >> filename;
    inFile.open(filename);
    do
    {
        count++;
    } while (NewLine (inFile));
    cout << "Number of lines:" << count << endl;
    return0;
}

bool NewLine (ifstream & in)
{
    char sym;
    bool flag = true;
    while (! in.eof() && flag)
    {
        in.get(sym);
        if (sym== '\n')
            flag = false;
    }
    return (!in.eof( ));
}
```

END OF LECTURE **********

MAY 10, 2000

last day - talked about eof( ) method and other file stuff
today - will talk about some functions useful for manipulating characters
will also talk about inheritance and structs.

Character manipulating functions
toupper - converts a character to upper case
e.g. cout << char (toupper ('a'));
outputs: A
char( ) is a function which converts an integer to a character to be printed.
toupper outputs an ASCII number, for an uppercase letter.

tolower - outputs lower case letter of a given upper case one.
isupper - returns true if character is an upper case letter.
islower - returns true if character is lower case
isalpha - true if character is a letter
isdigit - true if character is a number
isspace - true if character is a space

inheritance
recall classes are data types built out of smaller/simpler datatypes.
we can also use classes we've already defined to build new classes.
there are two ways to do this:
1. use the existing class like any other datatype in a class definition.
2. inheritance

input/ouput classes we've been using give an example of inheritance.
consider the following piece of code:

```
void two_sum (ifstream& sourceFile)
{
    int  n1, n2;
    sourceFile >> n1 >> n2;
    cout << n1 << "+" << n2 << "=" << (n1+n2) << endl;
}
```

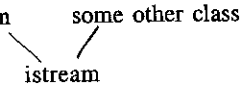
if we do:

```
ifstream fin;
fin.open ("file.dat")
two_sum (fin);
fin.close( );        //takes two int from file.dat and outputs sum to screen
```

consider:

```
two_sum(cin);     // does not work
```

would guess that this takes two numbers from keyboard and outputs sum to screen. Howerever, cin is an object of type istream (not of type ifstream), and so the type doesn't match. Nevertheless, ifstream class is defined from istream class using inheritance. This means you can use an ifstream object wherever an istream object could be used. ifstream objects can do additional things (like eof method) so cannot use an istream whenever an ifstream is used.
Since istream doesn't do as much as ifstream, in general, it will take less memory.
more than one class may inherit from istream.



can rewrite two_sum to exploit inheritance
if we did:

```
void better_two_sum (istream& sourceFile)
{
    //same code as before
}
```

then both

```
ifstream fin;
        fin.open ("file.dat");
        better_two_sum(fin);
        fin.close( );
```

And

```
better_two_sum(cin);
```

would work since fin is of type ifstream and so also of type istream. And cin is of type istream.

Some definitions:
ifstream is said to be a derived (or child) class of istream.
istream is said to be parent of ifstream.

e.g.

```
ofstream is derived from ostream
void Hello (ostream & out=cout)
{
    out << "hello\n";        out = cout means if no argument supplied, output to screen.
}

ofstream fout;
fout.open("file.dat");
Hello( );          //writes hello to screen.
Hello(cout);       //writes hello to screen.
Hello(fout);       //writes hello to file.dat
fout.close( );
```

Chapter 6: classes, struct, arrays

back in C days, did not have classes but C did have something called a struct
struct is like a class, but no inheritance, and no methods.
C++ is backward compatible (mainly), so has structs.
e.g.

```
struct Fraction
{
    int numerator;
    int denominator;
};
```

to use struct, could do

```
Fraction frac1;
```

creates one object of type Fraction.

To give it a value, could do

```
frac1.numerator = 3;
frac2.denominator=5;       //stores 3/5 in frac1
```

can use this type now in function definitions.

Fraction FracMult(Fraction& frac1, Fraction& frac2)

END OF LECTURE **********

MAY 12, 2000

structure
used for gluing together related data

e.g.
employee data
- first name
- last name
- salary
- empID

Employee structure definition

struct Employee

```
{
    char fname[20];
    char lname[20];      } member variables
    double salary;
    int EmpID;
};
```

can declare variables of type Employee.
in main, could do

```
Employee Fred;
```

To access member variables:

```
    Fred.salary
      ↑ ↑ ↑
name of | name of member variable
struct  dot operator
    variable of type double
```

```
ifstream in;
in.open("input.dat");
  ↑
dot operator
```

To initialize after declaration:

```
Employee Fred;
Fred.fname = "Fred";
Fred.lname = "Flintstone";
Fred.salary = 2;
Fred.EmpID = 123;
```

Just like:

```
int i;
i = 17;
```

To initialize in declaration:

```
    Employee Fred = {"Fred", "Flintstone", 2, 123};
                          ↑
```

List values in order of struct definition, separate with commas.

Just like:

```
int i = 17;
```

using member variables:

```
if (Fred.salary > 1000)
{
        // fire him.
}
```

using structs in functions:
can use any defined struct as a parameter. - must always be a call-by-reference parameter.

e.g. function to raise salary

```
void giveRaise (Employee& e, double raiseFactor)
{
    e.salary* = raiseFactor;
}       ↑
    using call-by-reference
```

```
Fred.salary = 2;
giveRaise (Fred, 0.75);
cout << "Fred's salary is" << Fred.salary << endl;
```

output:
Fred's salary is 1.5
(Notice that salary was decreased: can rename function to changeSalary, or design function such that a raise is guaranteed)

Arrays

what if you have 1000 employees?
could do
Employee emp0, emp1, emp2, ... emp998, emp999;

An array is a list

```
Employee emps[1000];      //creates 1000 Employee structures called emps[0],
                          // emps[1], ... , emps[999]
```

To access 10th employee's employee ID#

```
emps[9].EmpID
      ↑
```

because starts at 0
in the brackets, doesn't have to be an int, could be an arithmetic expression which evaluates to an int.

```
//give all even numbered employees a raise.
int i;
Employee emps[1000];
i = 0;
while (i < 500)
{
    emps[2*i].salary *=2;
    i++;
}
```

if you try to access emps[1000], may crash the program/computer.
compiler may or may not check for this.
same for emps[1001], emps[1002], ... emps[any number ≥ 1000]
→ index out of bounds → crash

```
struct Employee {
                                // have to define struct before using it in a prototype.
};
giveRaise (employee& e, double f);
```

END OF LECTURE AND SET #6 **********

PROGRAMMING IN COMPUTING 10A
PROFESSOR POLLETT
SET #7

MAY 15, 2000

Last day- talked about structs
Today- finish structs and begin talking about classes.

```
struct Employee
{
    char fname[20];
    char lname[20];
    double salary;
    int empID;
};
```

Once you create a struct, can use in definitions of new structs.

```
struct Dept
{
    Employee emp[1000]; // Notice we use things of type Employee.
    char deptName[20];
    char deptSupervisor[20];
};  // need semicolon.

Dept math;
// To print employee 10's first name in math Dept. would do
            i.e. the 11th employee
    cout <<math.emp[10].fname  << endl;
```

Classes

A class is a datatype whose variables are objects.
An object is a variable that has member functions as well as the ability to hold data.

e.g.

```
    # include <iostream.h>
    class Rational
    { public: // methods and data available to other classes and functions.
        void display ( );    // prototype for a function to display a rational
        int denominator;
        int numerator;
    };

    int main ( )
    {
     Rational a;
       cout << "Enter a numerator:";
       cin >> a.numerator;
       cout << "Enter a denominator:";
       cin >> a.denominator;
       cout << "The fraction you just input was";
       a.display ();
       return 0;
     }
```

```
void Rational : : display ( )
        // scope resolution operator. Like dot operator except rather than an
        // object name before it, : : has name of a class ( or a namespace)
{
    cout << numerator  << " / "  << denominator    << endl;
}
```

Output:
Enter a numerator: 5
Enter a denominator: 7
The fraction you just input was 5/7

The variables numerator and denominator in Rational : : display refer to the numerator and denominator of the current object under consideration. In a.display ( ); they would be a.numerator, a.denominator
Sometimes people use "*this" keyword.
i.e., could have written above method as

```
void Rational : : display ( )
{
    cout  << (*this).numerator  << " / "  << (*this).denominator  << endl;
}   // does same thing
```

Encapsulation: combining a number of items such as variables and functions into a single package, i.e. class.
Reason why we are doing this is to
1) improve organization of our code.
2) for information hiding.

Let's look at 2
Would like people to be able to use a class without needing to know how the data is stored. Thus, if people figure out a better way to store data, and use it in algorithm, the only code that needs to be changed is the class, and there is no need to change the code that uses the class.

e.g. could have written Rational as

```
class Rational
{ public:
    void set (int n, int d);
    int getNumerator ( ) ;
    int getDenominator ( );
    void display ( );
    // using these methods, a user could manipulate a rational.
private:
    int numerator;
    int denominator;
    // thsi data only available within class ( or to friend classes)
};
void Rational : : set (int n, int d)
{
    numerator = n;
    denominator = d;
}
int Rational : : getNumerator ( )
{
    return numerator;
}
int Rational : : getDenominator ( )
{
    return denominator;
}
```

accessor functions

END OF LECTURE **********

MAY 17, 2000

Last day - was talking about structs and classes
Today - will talk about equality for classes, and constructors, start talking about ADT's

Equality for objects
How to tell if two objects are qual
consider

```
class MyClass
{
    int member1;
    int member2;
};

int main( )
{
    MyClass a,b;
    a.member1=2;
    a.member2=4;
    b.member1=2;
    b.member2=4;
    if(a==b) cout << "hi\n";
    a = b;
    if (a==b) cout << "hi2\n";
    return 0;
}
```

output:
hi2

does not print hi even though a and b have the same internal values at first if statement.
why?
Because C++ will give an error if you have not specified what == does for a type.

Constructors:
idea: often you want to initialize an object when you declare it.
for variables of int's, double's, char's etc, can do things like:

```
int i = 7, b(6);
          ↑
          same as b = 6
```

let's see how to do this for classes

e.g.

```
class Rational
{
    public:
        Rational(int n, int d);      // constructor
            ↑                        // says how to make an object of type Rational
        notice no return type        // given 2 int's
        and name is same as class name

        Rational( );        // this constructor will be used to create a default rational 1/1.
        // rest of accessor functions from last day.
    private:
        int numerator, denominator;
};

Rational::Rational (int n, int d)
{
    numerator = n;
    denominator = d;
}
```

```
Rational::Rational( )
{
    numerator=1;
    denominator=1;
}

// code for other methods
```

Now we can do the following to create instances of Rational

```
Rational a = Rational(4, 5), b(2, 3), c, d;
d = Rational (4, 5)
```

a will store 4/5.
b will store 2/3.
c will store 1/1. uses constructor with arguments
d will store 4/5. d first created as 1/1, then created again as 4/5

Things you cannot do with constructors.

```
Rational e( );
          ↑
```

will cause an error. Should just do Rational e; to use zero parameter constructor.

```
c.Rational(2, 3);  // illegal, can't use constructors like ordinary methods.
```

notice we had two constructors for the class Rational. Can view this as overloading.
Overloading is completely legal for methods of a class.

ADT's

A datatype consists of a collection of values together with a basic set of operations on those values. i.e., +, *, /, ==, etc.

An abstract data type is a datatype where the details of the implementation of the datatype are hidden from the user.

e.g. could define a class for polynomials
e.g. of polynomials: $3x + 6$, $x^2 + 4$
in general, $a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0$

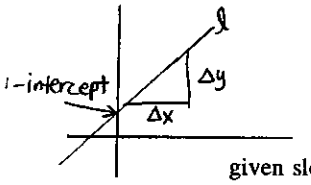
could define +, *, ==
for this class, two ways to store polynomials:
could store coefficients
e.g.
store (3, 6) for $3x + 6$ (coefficient representation)
or
could store enough points to fix polynomial
e.g. (0, 6)
(1, 9) for $3x + 6$ (point representation)
user doesn't need to know how the polynomials are stored for ADT.

END OF LECTURE **********

MAY 17, 2000

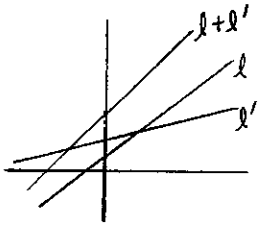
Last day - talked about classes. Started talking about ADT. Gave an example (without code) of polynomial ADT.
Today - keep talking about ADT's. Talk about Line class. (simplified variant of class Polynomial). Will talk about operator overloading and friend.

What is a Line?



$$\text{Slope} = \frac{\Delta y}{\Delta x}$$

given slope and y-intercept, can find any point on the line
→ (interpolation)



can add lines pointwise and get a new line

Often, we want to draw lines on screen or to a printer.
So it is useful to have a class Line.
We can view Line as an ADT and we have a choice of how to represent data internally. i.e., either as a slope and y-intercept, or as a pair of points.

Let's look at a C++ version of Line ADT for first representation.

```
class Line
{
    public:
            { Line (double m, double d);
constructors{ Line (double x0, double y0, double x1, double y1);
accessor    { double GetSlope ( );
function    { double GetYinter ( );
              double Interpolate (double x);
              friend Line operator + (const Line& l1, const Line& l2);
                  // this is a function that is defined outside of this class.
                  // friend means this function can view the private members of Line.
                  // Class objects used as function parameters must be passed call-by-reference. Const keyword says that despite
                    // this, the function will not be allowed to change the member values of this parameter. This function defines how + works
                   // for lines.
              friend bool operator == (const Line& l1, const Line& l2);
    private:
              double slope, yInter;
};  // end class definition.
```

The public part of above is called an interface. It doesn't change when you change internal representation.

```
Line::Line(double m, double b)
{
    slope = m;
    yInter = b;
}

Line::Line(double x0, double y0, double x1, double y1)
{
    slope = (y1-y0)/(x1-x0);
    yInter = y0-slope*x0;          // (y - y0) = m(0 - x0)
}

double Line::GetSlope ( )
{
    return slope;
}
```

y-intercept ↑

```
double Line::GetyInter ( )
{
    return yInter;
}


double Line::Interpolate (double x)
{
    return (slope*x + yInter);
}

Line operator+ (const Line& l1, const Line&l2)
{
    Line tmp(l1.slope + l2.slope, l1.yInter + l2.yInter);
    return tmp;
}

bool operator == (const Line& l1, const Line&2)
{
    return (l1.slope == l2.slope && l1.yInter == l2.yInter);
}
```

How you could use this class.

```
Line l, ll, lll;
l = Line(1,0);
ll = Line(0,0,1,1);
if (l == ll)
    cout << "hi\n";          // prints hi
lll = l + ll;
    cout << lll.GetSlope ( );      // prints 2.
```

What about if we used the other representation?

```
class Line
{
    // interface same as before
    private:
        double xzero, yzero, xone, yone;
};
    // end class definition.
```

Now we'd rewrite each method.
For example,

```
Line::Line (double m, double b)
{
    xzero = 0;
    yzero = b;
    xone = 1;
    yone = m + b;
}

Line::Line (double x0, double y0, double x1, double y1)
{
    xzero = x0;
    yzero = y0;
    xone = x1;
    yone = y1;
}
```

other methods done similarly.

END OF LECTURE AND SET #7 **********

LectureNotes

Spring 2000

PROGRAM IN COMPUTING 10A, S. 3 & 4
PROFESSOR POLLETT
SET #8

MAY 22, 2000

Last day - talked about classes, ADT's, operator overloading
Today - start Chapter 7, already seen lots of Ch 7 so will only cover new stuff. Today talk about union types, enum's, and switch/case.

Union Type

consider

```
struct Entry
{
    char name [20];
    int type; //holds which type to use of next two
    int int_value;
    double double_value;
};
```

In the way we will use this struct we are never going to use both int_value and double_value. So this struct wastes memory since we set aside enough memory for both values. How to avoid this? Use a union type.

So rewrite struct as:

```
struct Entry
{
        char name[20];
        int type;
        union {
            int int_value;
            double double_value;
            }
};
```

Union sets aside enough memory for the largest type appearing in it's block. In this case, set aside enough memory for one double.

e.g.

```
Entry a, b;
a.int_value = 10;
b.double_value = 4;
if(a.type = =1 && a.int_value = = 4) do_something( );
```

If store something in int_value but try to look at double_value (or vice versa), will get garbage.

Enumerated Types

Useful for making lists of declared constants.

e.g.

```
enum Direction { NORTH, SOUTH, EAST, WEST};  //have to have ;
```

This has roughly the same effect as

```
const int NORTH = 0, SOUTH = 1, EAST = 2, WEST = 3;
```

However, can create variahles of type Direction, like

```
Direction a;
```

You could imagine using this for controlling direction of a player in a game.
NORTH, SOUTH, EAST, WEST are easier to understand than 0, 1, 2, 3 which you may be using to control direction.
If using a number key pad, might want NORTH = 2, SOUTH = 8, etc.

To do this, could do:

```
enum Direction {NORTH = 2, SOUTH = 8, EAST = 6, WEST = 4};
```

Note values in an enum must be int's.
Once enum defined, can use
e.g. if(move = = NORTH)

```
MovePosition(NORTH, speed);
```

Notice that we always capitalized all letters in variables in enum. This is a common naming convention.

Switch/Case

Can use switch case to select from a list of possible things to do.
e.g.

```
switch(move)
        ↑
    must be an int or char
{
    case NORTH:
        MoveNorth( );
        break;
    case SOUTH;
        MoveSouth( );
        break
}
```

A switch statement uses its switch variable to select which case to execute.
Then continue to execute all statements from that case until a break statement is seen or switch ends.

```
char c;
cout << "Enter a grade:";
cin >> c;
switch (c)
{
    case 'A':
        cout << "Excellent\n";
        break;
    case 'B'
        cout << "Not bad\n";
        break;
    case 'C':
        cout << "OK\n";
        break;
    case 'D':
    case 'F':
        cout << "Tsk Tsk\n";
        break;
    default:
        cout << "Not a grade";
}
```

Output

```
Enter a grade: A
Excellent

Enter a grade: D
Tsk Tsk

Enter a grade: G
Not a grade
```

Switch/case usually compiles to faster code than equivalent if/else if have lots of cases.
Also, can make code more comprehensible.
default case is executed if switch variable doesn't match any of the cases.
default case is optional. If there is no default case and no match for switch variable, switch does nothing.

Frequently, use switch with menus.
For instance,

```
switch (GetChoice(1,4))
{
    case 1:
        CreateTable( );
        break;
    case 2:
        AddRow( );
        break;
    case 3:
        PrintTable( );
        break;
    case 4:
        exit(0);
}
```

END OF LECTURE **********

MAY 24, 2000

Last Day - union, enum, switch
Today - will talk a little about assignment in c++, then talk about for loops.
Blocks - the book makes a distinction between blocks and compound statements.
It calls a block a compound statement which initializes a variable
e.g.

```
int main ( )
{
    int a = 5;
    {
        int a = 6;
        cout << a << endl;
    }
    cout << a << endl;
}
```

block

Output:
6
5

Assignment
In C++, when you do an assignment like x = blah, the value x is first changed to blah, then the value of x is returned for use.

consider:

```
if (x=1)                    // this sets x to 1, then 1 is returned and used
    cout << "yep\n";        // by conditional. since 1 ≠ 0 = false, the conditional is true, so yep is printed
```

x++ is an assignment
we can use it in conditions
the semantics are not exactly like above.
e.g.

```
int x = 10;
do {
    cout << x << endl;
} while (x--);
```

↑
means: use value of x, then subtract 1.
output:
10
9
8
7
6
5

```
4
3
2
1
0
```

if you want something with the semantics of before, could do --x.

```
do {
        cout << x << endl;
} while (--x);
        ↑
        subtract one first, then use the value of x.
```

output:
```
10
9
8
7
6
5
4
3
2
1
```

e.g.
```
int x = 4, y = 4;
int v = x++;
int w = ++y;
cout << "x: " << x << endl
     << "v: " << v << endl
     << "y: " << y << endl
     << "w: " << w << endl;
```

output:
```
x:5
v:4
y:5
w:5
```

<u>for loops</u>

loops used for counting up/down through a set of values.
e.g.

initialization part (can create variables here) — loop condition — step

```
for  (int i=0; i<10; i++)
{
     cout << i << endl;
}
```

output:
```
0
1
2
3
4
5
6
7
```

8
9

so for loop works by first initializing counters. Then, it checks the loop condition, then executes the simple/compound statement after for loop. Then it does step phase. Rechecks condition, if it still holds, reexecute statement. Otherwise exit the loop and continue after for loop.

e.g.

```
int j, i;          <-- can have more than one thing if separated by a comma
for (i=0, j=5; i<j && i<10; j++, i+=3)
{
cout << "i: " << i << " "
     << "j: " << j << endl;
}
```

output:
i:0 j:5
i:3 j:6
i:6 j:7

since i,j are declared before the for loop, will still exist after it. So if we had cout << "i: " << i << " " << "j: " << j << endl;
right after for loop, would get
i:9 j:8
can omit any of the 3 components of for loops
e.g.
bad code but works

```
int i=0;
for (; i < 10; i++)
     cout << i << endl;
```

output:
0
1
2
.
.
.
9

```
for (int i=0; ; i++)
     cout << i << endl;
```

this is an infinite loop, prints:
0
1
2
3
.
.
.

Break
we've already seen break statements for switch statements
The meaning of break in a switch:

```
switch (var)
{
     case 10:
     break;
}
```

transfer control to after switch

The meaning of break in a loop is to send it to the first statement after the loop. For a switch, it is to send it to the first statement after switch.

```
int a;
for ( ; ; )                    // not from code in real life
{
    while (true)
    {
        cin >> a;
        if (a==7) break;
    }
    cout << "type 6 to quit\n";
    cin >> a;
    if (a==6) break;
}
```

output:
5
7
type 6 to quit
4
7
type 6 to quit
6

END OF LECTURE **********

MAY 26, 2000

Last day - was talking about for loops and break
Today - going to talk about goto, continue, arrays

Although we've never explicitly said it, it is completely legal in C++ to nest loops.

e.g.

```
while (cond1)
{
    for (init; cond2; step) ——— these loops are nested
    { if (cond3) break;        ← this break would transfer control to right after for loop
    }
}
```

Sometimes (say when reading data for a matrix into an array and suddenly get a file error) it is useful to break out of all loops in one go. We can use goto to do this.

e.g.

```
while (cond1)
{
    while (cond2)
    {
        if (in.eof( ))
        {
            cout<<"file ended early.\n;
            goto bob;
        }
    }
}

bob:    ← called a label, has no effect on execution
// rest of program.
```

If an eof is seen then control will be transferred to after bob. (After printing file ended early). Otherwise, two loops will execute to completion.

Continue statements

e.g.

```
struct Emp
{
    char name [20];
    double salary;
    bool keyToToilet;
}

Emp ACME[1000];
//Suppose we want to give everyone whose name doesn't begin with bob a raise and keys to toilet
for (int i =0; i<1000; i++)
{
    if (ACME [i].name[0] == 'b'&& ACME[i].name[1] =='o' && ACME[i].name[2]=='b')
        continue;
    ACME[i].salary*=1.1;
    ACME[i].keyToToilet = true;
}
```

continue causes us to skip over the rest of the loop ad go back to loop condition so if we had 3 employees anne, bob, and frank stored in ACME[0], ACME[1], ACME[2], then anne and frank would get raises and keys.

Arrays (Chapter 9)

Used to process a collection of data which are all of the same type.

E.g.

```
double scores[120];          //sets aside enough memory to store 120 doubles and give this memory the name scores.
```

To set value of an array element (say element 10), can do scores [10]=5;

To read an element, can do things like

```
if (scores[3]==10)
    cout <<"3 didn't do so well.\n";
```

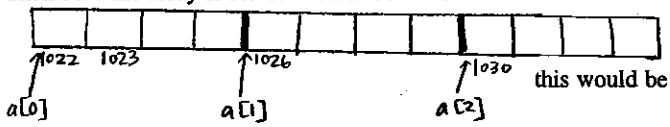
The range of the index for this array is between 0 and 119.

How does the computer create an array?

```
int a[3];       //an int is 4 bytes
```

suppose a[0] is stored at 1022, 1023, 1024, 1025. (Memory location).

Then bow the array is stored would look like



this would be the whole array.

When computer tries to figure out what/where a[i] is, it uses the formula (location of a[0] + i*(size of int))

If i is a variable, computer is not likely to catch if i is not 0, 1, 2. So if we write to or read from i outside of this range, it's quite likely that an error can occur, called index out of bounds error.

Some ways to initialize arrays that are more convenient

e.g.

```
if do ...
int a[3] = {8, 3, 1};
has same effect as...
int a[3];
a[0]=8;
a[1]=3;
a[2]=1;
can simplify this even more
    int a[ ] = {8, 3, 1};
          ↑
          compiler figures out the 3.
```

Arrays and functions

Have different cases...

If just want to pass a single element of an array, then it's easy.

E.g.

```
const int SIZE =3;
int abs (int j)
{
    if(j<0) j = -j;
    return j;
}
int main ( )
{
    int a[ ] = {1, -2, 3};
    for (int i=0; i<SIZE; i++)
    {
        cout <<abs(a[i]);
    }                    ↑
    return 0;        just like passing any variable to a function
}
```

output:

1
2
3

END OF LECTURE AND SET #8 **********

PROGRAM IN COMPUTING 10A
PROFESSOR POLLETT
SET #9

MAY 31, 2000

last day - started talking about passing arrays to functions
today - keep talking on the same topics, and talk about sorting

passing an element of an array to a function
e.g.

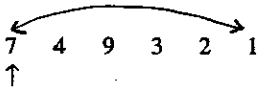
```
fun_test (a[4]);
          ↑
          passes element 5 of a to fun_test
```

today we'll show how to pass whole arrays to functions in the context of sorting

What is sorting?
Suppose you have an array of int's. To sort them means to place them in order from least to greatest.

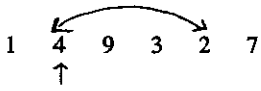
```
int a[ ] = {4, 1, 10};
    sorted would have
        a[0] = 1
        a[1] = 4
        a[2] = 10
```

e.g.
SelectionSort

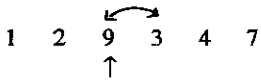


idea: go through array, swapping the value at the current index with the least value (including current index) of the values to its right.

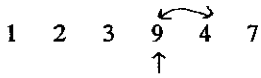
2nd step



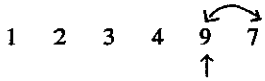
3rd step



4th step



5th step



6th step

1 2 3 4 7 9

Let's try to code this:

```
#include <iostream.h>
void Swap (int& one, int& two);
int FindMin (const int a[ ], int lo, int hi);
// finds min, index of least element between a[lo] and a[hi-1].
// const means this function should not change a[ ]
// int a[ ] means passing the whole array.  Arrays (like classes) are passed by reference.

void SelectionSort (int a[ ], int len)
// len is the length of a or more precisely, the number of elements of a we will sort.

void PrintOut (const int a[ ], int len);
// prints out a.

int main( )
{
    int test[ ] = {6, 2, 9};
    PrintOut (test, 3);
    // notice when passing array, no [ ]
    SelectionSort (test, 3);
    cout << "After sorting, \n";
    PrintOut (test, 3);
    return 0;
}

void PrintOut (const int a[ ], int len)
{
    for (int i = 0; i < len; i++)
    {
        cout << a[i] << "\t";
    }
    cout << endl;
}

void Swap (int& one, int& two)
{
    int tmp = one;
    one = two;
    two = tmp;
}

int FindMin (const int a[ ], int lo, int hi)
{
    int min = lo, minValue = a[lo];
    for (int i = lo; i < hi; i++)
    {
        if (minValue > a[i])
        {
            min = i;
            minValue = a[i];
        }
    }
    return min;
}

void SelectionSort (int a[ ], int len)
{
    for (int i=0; i < len; i++)
    {
        Swap (a[i], a[FindMin(a, i, len)]);
    }
}
```

returns int

Output:
6 2 9
After sorting,
2 6 9

how could we change the above to do sorting by strings?
Use alphabetical order
(lexicographical order)
e.g. ant > aadvark
lex
There are three useful C functions which will aid us in our quest.
strcmp(s, s2): compares s and s2, if s > s2, returns >0.
if s==s2, returns 0.
if s < s2, returns <0.

Another thing we'll need to change is Swap.
Will need some ability to copy strings.
strcpy(s, s2): copies string s2 into string s.

END OF LECTURE **********

JUNE 2, 2000

Last Day - was talking about SelectionSort and how to pass arrays to functions.
Today - how to modify SelectionSort to work with strings. Talk about multidimensional arrays, and pointers.

Rewriting some of the functions in SelectionSort for strings.

```
#include <cstring>      //notice no .h
                        //means same thing as string.h but emphasizes the
                        //fact that this is a C library as opposed to a C++
                        //library.

char names [3][20];
//creates a 3 x 20 array.  Can think of as 3 names of at most 20 char's.
//can assign names as names[0] = "Jeff";
//names[1] = "Bob";
//names[2] = "Sally";
//or could have assigned in one go:
//char names[3][20] = {"Jeff", "Bob", "Sally"};
//names[1][2] is b.(i.e. first name, second char).
//names[1][0] is B.
//names[1][1] is 0.
//names[1][3] is \0.  (end of string character).
//names[1][x] for x>3 is garbage.

//will rewrite FindMin.
    int FindMin (const char a[ ][20], int lo, int hi)
//when passing multidimensional arrays, have [ ] for 1st brackets,
//then put values in for sizes of remaining dimensions.
{
    int min = lo;
    char minValue[20];
    strcpy(minValue, a[lo]);        //copies string in a[lo] into minValue.
    for (int i=lo; i<hi; i++)
    {
        if(strcmp (minValue, a[i]>0)//strcmp is in cstring, returns > 0
                                    //if minValue > a[i] alphabetically
                                    //and returns 0 if same, and
                                    //returns < 0 if minValue < a[i]
        {
            min = i;
            strcpy(minValue, a[i]);
        } //end if
```

passes whole array minValue
passes whole row out of 2D array

```
    } //end for
    return min;
} //end of FindMin
```

e.g. How to call FindMin:

```
    int b = FindMin (names, 0, 3);
        this would set b equal to 1.
```

Some other useful functions of cstring.

strlen(s); → returns length of string s.

strcat(target_s, append_s); → sticks string append_s onto end of string target_s.
so if target_s = "hi" and append_s = "there" then afterwards target_s = "hithere".

Remark: could rewrite other functions from SelectionSort similar to above.

One last example before pointers.

You can of course have multidimensional arrays of things other than char's.

```
    int a[2][2] = {{0, 1}, {0, 2}};
        //this creates a 2 x 2 array with rows {0, 1} and {0, 2}.
    so value of a[0][1] is 1.
```

Pointers (Chap 11)
(sometimes called references)
these are variables used to store addresses in memory of where some type is stored.

e.g. `int *p, *q, r = 5;`

```
        ↑  ↑
        these are pointers to integers (store address in memory of an integer)
        p = &r;
            & before a variable returns its address.
    so p is the location in memory of where r is stored.
```

```
cout << p << endl;
//would print this address to screen.
cout << *p << endl;
    called dereferencing pointer p
    returns value stored at location p.
    So prints 5 to screen.
*p = 6;   //changes value of what's stored at location p to 6.
cout << r << endl;
//prints 6 to screen.
```

END OF LECTURE AND SET #9 **********

PROGRAM IN COMPUTING 10A S. 3&4
PROFESSOR POLLETT
SET #10

JUNE 5, 2000

Last day - talked about 2D-arrays, started talking about pointers.
Today - more pointers, dynamic memory allocation.

Recall

```
int *p;          //declares a pointer to an int
                 //i.e. p is the address in memory of where an
                 //integer is stored.
int v = 6;
p = &v;          //ampersand before variable returns the address of
                 //the variable. So p is now the address of v.
cout << p << endl;      //would print the memory location of v to the
                        //screen.
```

To get value of what a pointer p points to, you put a * in front of it.

```
cout << *p << endl;     //prints 6 to screen.
*p = 7;
cout << v << endl;      //prints 7 since above assignment changes
                        //what v stores.
```

What good are pointers?
One use is dynamic memory allocation. That is, memory we allocate while the program is running. Memory is allocated from a global store of memory known as the heap.

e.g.

```
typedef double* DoublePtr;
//this line tells compiler that DoublePtr is an abbreviation for double*.
//Now can do
DoublePtr myVar;
//same as if had done double *myVar;

//Now if do
myVar = new double;
//tells computer to allocate from heap enough money to
//store a double, then return the address of this memory and assign
//it to myVar.
*myVar = 6.7;     //stores 6.7 in this newly allocated memory.

//Once done using this memory, should return it so others can use it.
//To do this...
delete myVar;
//the memory myVar points to is now free for reuse.
```

One use for dynamic allocation is reading a file from a disk. Don't know how big the files will be at compile-time. At run time, can determine the file size, then dynamically allocate an array to hold its data, then read in file.

Dynamic allocation for 1-D arrays

e.g.

```
#include <iostream.h>
```

```
int main( )
{
    char*a;             //will be an array.
    int size;           //will get size of array.
    cout << "Enter an array size";
    cin >> size;
    a = new char[size];
    //now a should point to address of where size many characters are
    //available for use.
    if (a==NULL)        //NULL is value returned if allocation failed.
    {
        cout << "Not enough memory\n";
        return 1;
    }
    //Once you have allocated a, can use it like an ordinary array
    a[2] = 6;
    cout << a[2] << endl;

    //Now we're done using that memory and want to return it.
    delete[ ] a;
    //frees up the array of memory a points to.

    return 0;
}
```

Sometimes useful to dynamically allocate objects.

```
 class    MyType
{ public:
        MyType( );
        MyType(int, int);
        ~ MyType( );          //destructor
         int ID;
        int get( );
             :
};

int main( )
{
    MyType *a, *b;

    a = new MyType;  //uses the MyType( ) constructor.
    b = new MyType(3,4); //uses MyType(int,int) constructor.

    (*a).ID = 7 ;  //sets a's ID to 7. Can use a → ID = 7 instead
                   //(means same thing).
    cout << a → get( ) <<endl;       //prints some int to screen.

//To free up an object, do
    delete a;           //if defined, this calls the destructor for object.
    delete b;
    return 0;
}
```

2-D array dynamic allocation

```
//To allocate
int ** matrix;
//matrix is a pointer to array of pointer for the rows. Each row is a
//pointer to an array of its columns.
//To allocate
int rows = 6, cols = 7;
matrix = new int*[rows];
    //allocates an array to store rows. Each row is a pointer to its columns.
```

```
for(int j = 0; j < row; j++)
{
    matrix[j] = new int[cols];
}
//above allocates memory for each row.
//Now can use like any 2D-array.
matrix[3][4] = 4;
cout << matrix [3][4] << endl;

//To deallocate
for (int i = 0; i < rows; i++)
{
    delete[ ] matrix[i];
}
//above gets rid of memory for rows.

delete[ ]matrix;
//frees matrix of rows.
```

END OF LECTURE **********

JUNE 7, 2000

last new topic: destructor
when an object of a class is deleted, its destructor is called. The destructor's job is to free up any memory that might have been dynamically allocated by that object. In HW6, its job is to free up the game board.

```
class MyType
{
    public:
        MyType( );          //constructor
        ~MyType( );         //destructor
                            //same name as class with ~ in front.

    private:
        int *p, q;
};

MyType::MyType( )
{
    q = 6;
    p = new int[10];        //dynamic memory allocation.
}
// if didn't have destructor and did MyType *obj;
//                              obj = new MyType;
//                              delete obj;
// computer would free up memory for q and where pointer p was stored but would not free up the 10 int's we've dynamically allocated.

// to write destructor
MyType::~MyType( )
{
    delete[ ] p;
}
// now all memories are reclaimed
```

<u>practice final</u>

1) define and explain
a) call-by-reference
calling a function with the address of a variable rather than its actual value. So if change value of variable in the function, this will affect the value in the calling function.

```
E.g. #include <iostream.h>
     void f(int& a);
```

```
int main( )
{
    int a=5;
    f(a);
    cout << a << endl;
    return 0;
}

void f(int& b)
{
    b++;
}
```

prints 6.

b) post-condition
a condition on what is returned by a function or a block; or else, a condition on the value of a variable after a function or block has run.

E.g.

```
int f( )
{
    return 6;
}
// a postcondition would be:
// returns value 6

int i=0;
while (i<10) i++;
// postcondition i==10
```

c) inheritance
a property of a class defined in terms of another class or classes by adding member variables/functions. The property is that the newly defined class has all the member functions/variables of the original class.

E.g. ifstream inherits from istream all the functions like setf, width, etc. of istream.
In addition, it has open, close which are new.

d) accessor function
a member function of a class that allows users of the class to access or set private member variables of that class.
E.g. setf in istream
is an accessor function which allows us to get private properties of an istream object.

e) friend function of a class is a function that is allowed to access the private variables/member functions of that class.

E.g.

```
class A
{
    public:
        A(int i=0);
        friend bool operator == (A& c, A& d);
    private:
        int myInt;
};

bool operator == (A& c, A& d)
{
    return c.myInt == d.myInt;
}

A::A (int i=0)
{
    myInt = i;
}
```

```
A f,g;
if (f==g) cout << "hi\n";          // prints hi
```

2)
```
#include <iostream.h>
int main ( )
{
    double i;
    for (i=0; i<=100; i=i+0.001)
    {
        cout.setf(ios::fixed);
        cout.setf(ios::showpoint);
        cout.setf(ios::left);
        cout.precision(3);
        cout.width(10);
        cout << i << endl;
    }
    return 0;
}
```

END OF LECTURE **********

JUNE 9, 2000

Practice Final (continued)

3)
```
#include <iostream.h>
#include <fstream.h>
int main()
{
    char name [20];   //stores filename
    int num;
    ofstream out;
    cout <<  "Enter a filename:";
    cin >> name;
    out.open (name);
    if (out.fail())
    {
        cout << "Error opening file\n";
        return 1;
    }
    cout << "Enter a number:";
    cin >> num;
    out << num;
    out.close();
    return 0;
}
```

4) Write member functions for the following class.

```
class Book
{ public:
    Book (char n[ ], bool stat);
    void getName (char copiedName[ ]);
    bool getCheckOutStatus ();
  private:
    char name[10];
    bool checkOutStatus;
};

Book :: Book (char n[ ], bool stat)
{
    strcpy (name, n);   // assume we had #include <string.h>
    checkOutStatus = stat;
}
```

```
void Book ::getName (char copiedName[ ])
{
    strcpy  (copiedName, name);
}

bool Book::getCheckOutStatus ()
{
    return checkOutStatus;
}
```

5) What does the following print?

```
int i,j;
for (i=0, j=3, ; i*j<5;  i++, j=(j +1) % 4)
{
    if (i==1)   continue;
    cout << "i:" << i <<  "j:" << j << endl;
}
```

output:
i:0 j:3
i:2 j:1

6)

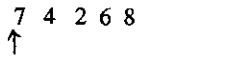
```
int GradesT ;GPA (char grade)
{
    switch (grade)
    {
        case 'A':
            return 4;
        case 'B':
            return 3;
        case 'C':
            return 2;
        case 'D':
            return 1;
    }
    return 0;
}
```

7) break -- flow of control after a break goes to immediately after nearest enclosing loop.
goto -- flow of control transfers immediately from goto statement to where its label is.
goto should only be used to break out of several nested loops all in one go.
e.g.

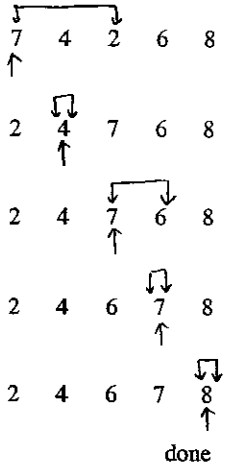
```
int num;
while (true)
{
    while (true)
    {
        cout << "Enter a number:";
        cin >> num;
        if (num==1) break;
        if (num==2)  goto bob;
        cout << "inner loop\n";
    } // end inner while loop.
    cout << "outer loop \n";
} // end outer loop.
bob:
cout << "done\n";
```

Output:
Enter a number: 3
inner loop
Enter a number: 1
outer loop
Enter a number: 2
done

8) Explain how selection sort sorts the following: 7 4 2 6 8



0) put an arrow under left most element
1) look at numbers to right of (and including) arrowed element;
swap arrowed element with smallest of these numbers.
2) move arrow right one element. If arrow still in list of elementss, go to step 1, else stop.



9) What does the following print out?

```
#include <iostream.h>
int *p, *q, v=7;
cout << "v:" << v << endl;
p=& v;
cout << "*p:" << *p << endl;
*p=1;
cout << "*p:" << *p << endl;
q=new int;
*q=4;
p=q;
cout << "*p:" << *p << endl;
```

Output:

```
v:7
*p:7
*p:1
*p:4
```

10)

```
#include <iostream.h>
int main ()
{
    char ** arr;
    int n, m, i;
    cout << "Enter two integers:";
    cin >> n >> m;
    arr  = new char * [n];
    for (i=0; i<n; i++)
    {
        arr[i] = new char [m];
    }
    for (i=0; i<n; i++)
    {
        cout << "Enter a string for row " << i << ": ";
        cin >> arr[i];
    }
    for (i=0; i<n; i++)
    {
        cout << arr[i] << endl;
    }
```

```
    for (i=0; i<n; i++)
    {
        delete [ ] arr[i];
    }
    delete [ ] arr;
    return 0;
}
```

END OF LECTURE & END OF SET # 10********