LITERALS

255, 0377, 0xff	*Integers (decimal, octal, hex)*
2147483647L, 0x7fffffffl	*Long (32-bit) integers*
123.0, 1.23e2	*double (real) numbers*
'a', '\141', '\x61'	*Character (literal, octal, hex)*
'\n', '\\', '\'', '\"'	*Newline, backslash, single quote, double quote*
"string\n"	*Array of characters ending with newline and \0*
"hello" "world"	*Concatenated strings*
true, false	*bool constants 1 and 0*

STORAGE CLASSES

int x;	*Auto (memory exists only while in scope)*
static int x;	*Global lifetime even if local scope*
extern int x;	*Information only, declared elsewhere*

C++ PROGRAM STRUCTURE

```
// my first program in C++
#include <iostream.h>
#define X \
    some text            // Line continuation
int main () {
    int x;            // Scope of x is from
                     // declaration to end of block
    cout << "Hello World!";   // Every expression
                              // is a statement
    return 0;
}
/* multi-line
   comment */
```

IDENTIFIERS

ANSI C++ reserved words, cannot be used as variable names.
asm, **auto**, **bool**, **break**, **case**, **catch**, **char**, class, **const**, **const_cast**, **continue**, **default**, **delete**, **do**, **double**, **dynamic_cast**, **else**, **enum**, **explicit**, **extern**, false, **float**, **for**, **friend**, **goto**, **if**, **inline**, **int**, **long**, **mutable**, **namespace**, **new**, **operator**, **private**, **protected**, **public**, **register**, **reinterpret_cast**, **return**, **short**, **signed**, **sizeof**, **static**, **static_cast**, **struct**, **switch**, **template**, **this**, **throw**, true, try, **typedef**, **typeid**, **typename**, **union**, **unsigned**, **using**, **virtual**, **void**, **volatile**, **wchar_t**

DATA TYPES

VARIABLE DECLARATION

```
special class size sign type name;
volatile                              // special
register, static, extern, auto        // class
long, short, double                   // size
signed, unsigned                      // sign
int, float, char                      // type (required)
the_variable_name                     // name (required)
// example of variable declaration
extern short unsigned char AFlag;
```

TYPE	SIZE	RANGE
char	1	*signed -128...127* *unsigned 0...255*
short	2	*signed -32768...32767* *unsigned 0...65535*
long	4	*signed -2147483648...2147483647* *unsigned 0...4294967295*
int		*varies depending on system*
float	4	*3.4E +/- 38 (7 digits)*
double	8	*1.7E +/- 308 (15 digits)*
long double	10	*1.2E +/- 4932 (19 digits)*
bool	1	*true or false*
wchar_t	2	*wide characters*

POINTERS

```
type *variable;  // pointer to variable
type *func();    // function returns pointer
void *  // generic pointer type
NULL;    // null pointer
*ptr;    // object pointed to by pointer
&obj;    // address of object
```

ARRAYS

```
int arry[n];          // array of size n
int arry2d[n][m];     // 2d n x m array
int arry3d[i][j][k]; // 3d i x j x k array
```

STRUCTURES

```
struct name {
    type1 element1;
    type2 element2;
    int   anum;
    char  achar;
} object_name;       // instance of name
name variable;       // variable of type name
variable.element1;  // ref. of element
variable->element1; // reference of pointed to structure
```

INITIALIZATION

```
type id;            // declaration
type id,id,id;      // multiple declaration
type *id;           // pointer declaration
type id = value;    // declare with assign
type *id = value;   // pointer with assign
id = value;         // assignment
```

EXAMPLES

```
char c='A'; // single character in single quotes
char *str = "Hello"; // string in double quotes,
                     // pointer to string
int i = 1022;
float f = 4.0E10;     // 4^10
int a[10];            // Array of 10 ints a[0]-a[9]
int ary[2] = {1,2};   // Array of ints
int a[]={0,1,2};      // Initialized array  a[3]={0,1,2};
int a[2][3]={{1,2,3},{4,5,6}};  // Array of array of ints
const int a = 45;    // constant declaration
struct products {    // declaration
    char name [30];
    float price;
};
products apple;                 // create instance
apple.name = "Macintosh";       // assignment
apple.price = 0.45;
products *pApple;               // pointer to struct
pApple->name = "Granny Smith";
pApple->price = 0.35;           // assignment
short s; long l;      // Usually 16 or 32 bit integer
                      // (int may be either)
unsigned char u=255;
signed char s=-1;      // char might be either
unsigned long x=0xffffffffL; // short, int, long
                             // are signed
float f; double d; // Single or double precision real
                   // (never unsigned)
bool b=true;     // true or false, may use int (1 or 0)
int* p;            // p is a pointer to (address of) int
char* s="hello";   // s points to unnamed array
                   // containing "hello"
void* p=NULL;      // Address of untyped memory (NULL is 0)
int& r=x;          // r is a reference to (alias of) int x
enum weekend {SAT,SUN};  // weekend is a type with values
                         // SAT and SUN
enum weekend day;      // day is a variable of type weekend
enum weekend {SAT=0,SUN=1};// Explicit representation as int
enum {SAT,SUN} day;    // Anonymous enum
typedef String char*; // String s; means char* s;
const int c=3;          // Constants must be initialized,
                        // cannot assign to
const int* p=a;         // Contents of p (elements of a)
                       // are constant
int* const p=a;         // p (but not contents) are constant
const int* const p=a; // Both p and its contents
                        // are constant
const int& cr=x;        // cr cannot be assigned to change x
```

EXCEPTIONS

```
try {
    // code to be tried
    statements; // if statements fail, exception is set
    throw exception;
}
catch (type exception) {
    // code in case of exception
    statements;
}
```

USER DEFINED DATATYPES

```
typedef existingtype newtypename;
typedef unsigned int WORD;
enum name{val1, val2, ...} obj_name;
enum days_t {MON,WED,FRI} days;
```

```
union model_name {
    type1 element1;
    type2 element2;
    ...
} object_name ;
union mytypes_t {
    char c;
    int i;
} mytypes;
struct packed {                // bit fields
    unsigned int flagA:1;   // flagA is 1 bit
    unsigned int flagB:3;   // flagB is 3 bit
}
```

OPERATORS

priority/**operator**/desc/ASSOCIATIVITY

1	::	**scope**	LEFT
2	()	**parenthesis**	LEFT
	[]	**brackets**	LEFT
	->	**pointer reference**	LEFT
	.	**structure member access**	LEFT
	sizeof	**returns memory size**	LEFT
3	++	**increment**	RIGHT
	--	**decrement**	RIGHT
	~	**complement to one (bitwise)**	RIGHT
	!	**unary NOT**	RIGHT
	&	**reference (pointers)**	RIGHT
	*	**dereference**	RIGHT
	(type)	**type casting**	RIGHT
	+ -	**unary less sign**	RIGHT
4	*	**multiply**	LEFT
	/	**divide**	LEFT
	%	**modulus**	LEFT
5	+	**addition**	LEFT
	-	**subtraction**	LEFT
6	<<	**bitwise shift left**	LEFT
	>>	**bitwise shift right**	LEFT
7	<	**less than**	LEFT
	<=	**less than or equal**	LEFT
	>	**greater than**	LEFT
	>=	**greater than or equal**	LEFT
8	==	**equal**	LEFT
	!=	**not equal**	LEFT
9	&	**bitwise AND**	LEFT
	^	**bitwise NOT**	LEFT
	|	**bitwise OR**	LEFT
10	&&	**logical AND**	LEFT
	||	**logical OR**	LEFT
11	? :	**conditional**	RIGHT
12	=	**assignment**	
	+=	**add/assign**	
	-=	**subtract/assign**	
	*=	**multiply/assign**	
	/=	**divide/assign**	
	%=	**modulus/assign**	
	>>=	**bitwise shift right/assign**	
	<<=	**bitwise shift left/assign**	
	&=	**bitwise AND/assign**	
	^=	**bitwise NOT/assign**	
	|=	**bitwise OR/assign**	
13	,	**comma**	

PREPROCESSOR DIRECTIVES

```
#define ID value // replaces ID with
                 //value for each occurrence in the code
#undef ID    // reverse of #define
#ifdef ID    //executes code if ID defined
#ifndef ID   // opposite of #ifdef
#if expr     // executes if expr is true
#else        // else
#elif        // else if
#endif       // ends if block
#line number “filename” // #line controls what line number
                        // and filename appear when compiler
                        // error occurs
#error msg  //reports msg on cmpl. error
#include “file”  // inserts file into code
                 // during compilation
#pragma          //passes parameters to compiler
#include <stdio.h>  //  Insert standard header file
#include "myfile.h" //  Insert file in current directory
#define F(a,b) a+b  //  Replace F(1,2) with 1+2
#undef X         //  Remove definition
#if defined(X)   //  Condional compilation (#ifdef X)
#else            //  Optional (#ifndef X or #if !defined(X))
#endif           //  Required after #if, #ifdef
```

CONTROL STRUCTURES

DECISION (if-else)

```
if (condition) {
    statements;
}
else if (condition) {
    statements;
}
else {
    statements;
}
if (x == 3)      // curly braces not needed
    flag = 1;     // when if statement is
else             // followed by only one
    flag = 0;     // statement
```

REPETITION (while)

```
while (expression) {  // loop until
    statements;        // expression is false
}
```

REPETITION (do-while)

```
do {                  // perform the statements
    statements;        // as long as condition
} while (condition);  // is true
```

REPETITION (for)

init	initial value for loop control variable
condition	stay in the loop as long as condition is true
increment	change the loop control variable

```
for(init; condition; increment) {
    statements;
}
```

BIFURCATION (break, continue, goto, exit)

```
break;    // ends a loop
continue; // stops executing statements in current
          // iteration of loop continues executing
          // on next iteration
label:
goto label;    // execution continues at label
exit(retcode); // exits program
```

SELECTION (switch)

```
switch (variable) {
    case constant1: // chars, ints
        statements;
        break;       // needed to end flow
    case constant2:
        statements;
        break;
    default:
        statements; // default statements
}
```

CONSOLE I/O

C STYLE CONSOLE I/O

stdin	standard input stream
stdout	standard output stream
stderr	standard error stream

```
// print to screen with formatting
printf("format", arg1,arg2,...);
printf("nums: \%d, \%f,\%c", 1, 5.6, 'C');
// print to string s
sprintf(s, "format", arg1, arg2,...);
sprintf(s, "This is string # \%i",2);
```

```
// read data from keyboard into
// name1, name2,...
scanf("format", &name1, &name2, ...);
scanf("\%d,\%f", var1, var2); // read nums
// read from string s
sscanf("format", &name1, &name2, ...);
sscanf(s, "\%i,\%c", var1, var2);
```

C STYLE I/O FORMATTING

%d, %i	integer
%c	single character
%f	double (float)
%o	octal
%p	pointer
%u	unsigned
%s	char string
%e, %E	exponential
%x, %X	hexadecimal
%n	number of chars written
%g, %G	same as f for e,E

C++ CONSOLE I/O

cout<<	console out, printing to screen
cin>>	console in, reading from keyboard
cerr<<	console error
clog<<	console log

```
cout<<"Please enter an integer: ";
cin>>i;
cout<<"num1: "<<i<<"\n"<<endl;
```

CONTROL CHARACTERS

\b	backspace	\f	form feed
\r	return	\’	apostrophe
\n	newline	\t	tab
\nnn	character #nnn (octal)	\”	quote
\NN	character #NN (hexadecimal)		

CHARACTER STRINGS

The string “Hello” is actually composed of 6 characters and is stored in memory as follows:

Char:	H	e	l	l	o	\0
Index:	0	1	2	3	4	5

\0 (backslash zero) is the null terminator character and determines the end of the string. A string is an array of characters. Arrays in C and C++ start at zero.

```
str = "Hello";
str[2] = 'e'; // string is now ‘Heelo’
```

common <string.h> functions:

```
strcat(s1,s2) strchr(s1,c)     strcmp(s1,s2) strcpy(s2,s1)
strlen(s1)    strncpy(s2,s1,n) strstr(s1,s2)
```

FUNCTIONS

In C, functions must be prototyped before the main function, and defined after the main function. In C++, functions may, but do not need to be, prototyped. C++ functions must be defined before the location where they are called from.

```
// function declaration
type name(arg1, arg2, ...) {
    statement1;
    statement2;
    ...
}
```

type	return type of the function
name	name by which the function is called
arg1, arg2	parameters to the function
statement	statements inside the function

```
// example function declaration
// return type int
int add(int a, int b) {   // parms
    int r;                // declaration
    r = a + b;            // add nums
    return r;             // return value
}
num = add(1,2);           // function call
```

PASSING PARAMETERS

BY VALUE

```
function(int var); // passed by value
```

Variable is passed into the function and can be changed, but changes are not passed back.

BY CONSTANT VALUE

```
function(const int var); // passed by constant
```

Variable is passed into the function but cannot be changed.

BY REFERENCE

```
function(int &var); // pass by reference
```

Variable is passed into the function and can be changed, changes are passed back.

BY CONSTANT REFERENCE

```
function(const int &var);
```

Variable cannot be changed in the function.

ARRAY BY REFERENCE

It’s a waste of memory to pass arrays and structures by value, instead pass by reference.

```
int array[1];               // array declaration
ret = aryfunc(&array);     // function call
int aryfunc(int *array[1]) {
    array[0] = 2;          // function
    return 2;              // declaration
}
```

DEFAULT PARAMETER VALUES

```
int add(int a, int b=2) {
    int r;
    r = a + b;      // b is always 2
    return (r);
}
```

OVERLOADING FUNCTIONS

Functions can have the same name, and same number of parameters as long as the parameters are of different types

```
// takes and returns integers
int divide (int a, int b)
    { return (a/b); }
// takes and returns floats
float divide (float a, float b)
    { return (a/b); }
divide(10,2);    // returns 5
divide(10,3);    // returns 3.33333333
```

RECURSION

Functions can call themselves

```
long factorial (long n) {
    if (n > 1)
        return (n * factorial (n-1));
    else
        return (1);
}
```

PROTOTYPING

Functions can be prototyped so they can be used after being declared in any order

```
// prototyped functions can be used
// anywhere in the program
#include <iostream.h>
void odd (int a);
void even (int a);
int main () { ... }
```

NAMESPACES

Namespaces allow global identifiers under a name

```
// simple namespace
namespace identifier {
    namespace body;
}
// example namespace
namespace first {int var = 5;}
namespace second {double var = 3.1416;}
int main () {
    cout << first::var << endl;
    cout << second::var << endl;
    return 0;
}
// using namespace allows for the current nesting
// level to use the appropriate namespace
using namespace identifier;
// example using namespace
namespace first {int var = 5;}
namespace second {double var = 3.1416;}
int main () {
    using namespace second;
    cout << var << endl;
    cout << (var*2) << endl;
    return 0;
}
```

CLASS REFERENCE

CLASS SYNTAX

```
class classname {
    public:
        classname(parms);    // constructor
        ~classname();        // destructor
        member1;
        member2;
    protected:
        member3;
        ...
    private:
        member4;
} objectname;
// constructor (initializes variables)
classname::classname(parms) {
}
// destructor (deletes variables)
classname::~classname() {
}
```

public	members are accessible from anywhere where the class is visible
protected	members are only accessible from members of the same class or of a friend class
private	members are accessible from members of the same class, members of the derived classes and a friend class
constructors	may be overloaded just like any other function. define two identical constructors with difference parameter lists

CLASS EXAMPLE

```
class CSquare {                // class declaration
    public:
        void Init(float h, float w);
        float GetArea();      // functions
    private:                   // available only to CSquare
        float h,w;
}
// implementations of functions
void CSquare::Init(float hi, float wi){
    h = hi; w = wi;
}
float CSquare::GetArea() {
    return (h*w);
}
// example declaration and usage
CSquare theSquare;
theSquare.Init(8,5);
area = theSquare.GetArea();
// or using a pointer to the class
CSquare *theSquare;
theSquare->Init(8,5);
area = theSquare->GetArea();
```

OVERLOADING OPERATORS

Like functions, operators can be overloaded. Imagine you have a class that defines a square and you create two instances of the class. You can add the two objects together.

```
class CSquare {      // declare a class
    public:          // functions
        void Init(float h, float w);
        float GetArea();
        CSquare operator + (CSquare);
    private:        // overload the ‘+’ operator
        float h,w;
}
// function implementations
void CSquare::Init(float hi, float wi){
    h = hi; w = wi;
}
float CSquare::GetArea() {
    return (h*w);
}
// implementation of overloaded operator
CSquare CSquare::operator+ (CSquare cs) {
    CSquare temp;       // create CSquare object
    temp.h = h + cs.h;  // add h and w to
    temp.w = w + cs.w;  // temp object
    return (temp);
}
// object declaration and usage
CSquare sqr1, sqr2, sqr3;
sqr1.Init(3,4);            // initialize objects
sqr2.Init(2,3);
sqr3 = sqr1 + sqr2;        // object sqr3 is now (5,7)
```

ADVANCED CLASS SYNTAX

STATIC KEYWORD

static	variables are the same throughout all instances of a class.

```
static int n;    // declaration
CDummy::n;       // reference
```

VIRTUAL MEMBERS

Classes may have virtual members. If the function is redefined in an inherited class, the parent must have the word virtual in front of the function definition

THIS KEYWORD

The this keyword refers to the memory location of the current object.

```
int func(this);  // passes pointer to current object
```

CLASS TYPECASTING

```
reinterpret_cast  <newtype>(expression);
dynamic_cast       <newtype>(expression);
static_cast        <newtype>(expression);
const_cast        <newtype>(expression);
```

EXPRESSION TYPE

The type of an expression can be found using typeid.

```
typeid(expression);   // returns a type
```

INHERITANCE

Functions from a class can be inherited and reused in other classes. *Multiple inheritance* is possible.

```
class CPoly {       //create base polygon class
    protected:
        int width, height;
    public:
        void SetValues(int a, int b)
            { width=a; height=b;}
};
class COutput {     // create base output class
    public:
        void Output(int i);
};
void COutput::Output (int i) {
    cout << i << endl;
}
// CRect inherits SetValues from Cpoly
// and inherits Output from COutput
class CRect: public CPoly, public COutput
{
    public:
        int area(void)
            { return (width * height); }
};
// CTri inherits SetValues from CPoly
class CTri: public CPoly {
    public:
        int area(void)
            { return (width * height / 2); }
};
void main () {
    CRect rect;    // declare objects
    CTri tri;
    rect.SetValues (2,9);
    tri.SetValues (2,9);
    rect.Output(rect.area());
    cout<<tri.area()<<endl;
}
```

TEMPLATES

Templates allow functions and classes to be reused without overloading them

```
template <class id> function;
template <typename id> function;
// ---------- function example ---------
template <class T>
T GetMax (T a, T b) {
    return (a>b?a:b); // return the larger
}
void main () {
    int a=9, b=2, c;
    float x=5.3, y=3.2, z;
    c=GetMax(a,b);
    z=GetMax(x,y);
}
// ----------- class example -----------
template <class T>
class CPair {
    T x,y;
    public:
        Pair(T a, T b){
            x=a; y=b; }
        T GetMax();
};
template <class T>
T Pair<T>::GetMax()
{ // implementation of GetMax function
    T ret;          // return a template
    ret = x>y?x:y; // return larger
    return ret;
}
int main () {
    Pair <int> theMax (80, 45);
    cout << theMax.GetMax();
    return 0;
}
```

FRIEND CLASSES/FUNCTIONS

FRIEND CLASS EXAMPLE

```
class CSquare;        // define CSquare
class CRectangle {
    int width, height;
    public:
        void convert (CSquare a);
};
class CSquare {       // we want to use the
    private:          // convert function in
        int side;     // the CSquare class, so
    public:           // use the friend keyword
        void set_side (int a) { side=a; }
        friend class CRectangle;
};
void CRectangle::convert (CSquare a) {
    width = a.side;
    height = a.side;
}
// declaration and usage
CSquare sqr;
CRectangle rect;     // convert can be
sqr.set_side(4);     // used by the
rect.convert(sqr);   // rectangle class
```

FRIEND FUNCTIONS

A friend function has the keyword friend in front of it. If it is declared inside a class, that function can be called without reference from an object. An object may be passed to it.

```
/* change can be used anywhere and can have a CRect
object passed in */
// this example defined inside a class
friend CRect change(CRect);
CRectangle recta, rectb;    // declaration
rectb = change(recta);      // usage
```

FILE I/O

```
#include <fstream.h>    // read/write file
#include <ofstream.h>   // write file
#include <ifstream.h>   // read file
```

File I/O is done from the classes fstream, ofstream, ifstream.

FILE HANDLES

A file must have a file handle (pointer to the file) to access the file.

```
ifstream infile;     // create handle called
                     // infile to read from a file
ofstream outfile;    // handle for writing
fstream f;           // handle for read/write
```

OPENING FILES

After declaring a file handle, the following syntax can be used to open the file

```
void open(const char *fname, ios::mode);
```

fname	should be a string, specifying an absolute or relative path, including filename
ios::in	Open file for reading
ios::out	Open file for writing
ios::ate	Initial position: end of file
ios::app	Every output is appended at the end of file
ios::trunc	If the file already existed it is erased
ios::binary	Binary mode

```
ifstream f;             // open input file example
f.open("input.txt", ios::in);
ofstream f;             // open for writing in binary
f.open("out.txt", ios::out | ios::binary | ios::app);
```

CLOSING A FILE

```
f.close();    // close the file with handle f
```

WRITING TO A FILE (TEXT MODE)

The operator << can be used to write to a file. Like cout, a stream can be opened to a device. For file writing, the device is not the console, it is the file. cout is replaced with the file handle.

```
ofstream f; // create file handle
f.open("output.txt") // open file
f <<"Hello World\n"<<a<<b<<c<<endl;
```

READING FROM A FILE (TEXT MODE)

The operator >> can be used to read from a file. It works similar to cin. Fields are seperated in the file by spaces.

```
ifstream f;             // create file handle
f.open("input.txt");    // open file
while (!f.eof())        // end of file test
    f >>a>>b>>c;        // read into a,b,c
```

I/O STATE FLAGS

Flags are set if errors or other conditions occur. The following functions are members of the file object

```
handle.bad()    /* returns true if a failure occurs in
                   reading or writing */
handle.fail()   /* returns true for same cases as bad()
                   plus if formatting errors occur */
handle.eof()    /* returns true if the end of the file
                   reached when reading */
handle.good()   /* returns false if any of the above
                   were true */
```

STREAM POINTERS

```
handle.tellg()   // returns pointer to current location
                 // when reading a file
handle.tellp()   // returns pointer to current location
                 // when writing a file
// seek a position in reading a file
handle.seekg(position);
handle.seekg(offset, direction);
// seek a position in writing a file
handle.seekp(position);
handle.seekp(offset, direction);
```

direction can be one of the following

ios::beg	beginning of the stream
ios::cur	current position of the stream pointer
ios::end	end of the stream

BINARY FILES

buffer	a location to store the characters.
numbytes	the number of bytes to written or read.

```
write(char *buffer, numbytes);
read(char *buffer, numbytes);
```

OUTPUT FORMATTING

```
streamclass f;              // declare file handle
f.flags(ios_base::flag)     // set output flags
```

possible flags

dec	fixed	hex
oct	scientific	internal
left	right	uppercase
boolalpha	showbase	showpoint
showpos	skipws	unitbuf
adjustfield left	adjustfield right	adjustfield internal
basefield dec	basefield oct	basefield hex
floatfield scientific	floatfield fixed	

```
f.fill()      // get fill character
f.fill(c h)   // set fill character ch
f.precision(ndigits) // sets the precision for floating
                     // point numbers to ndigits
f.put(c)      // put a single char into output stream
f.setf(flag) // sets a flag
f.setf(flag, mask) // sets a flag w/value
f.width()     // returns the current number of characters
              // to be written
f.width(num) // sets the number of chars to be written
```

DYNAMIC MEMORY

Memory can be allocated and deallocated

```
// allocate memory (C++ only)
pointer = new type [];
int *ptr;               // declare a pointer
ptr = new int;          // create a new instance
ptr = new int [5];      // new array of ints
// deallocate memory (C++ only)
delete [] pointer;
delete ptr;             // delete a single int
delete [] ptr           // delete array
// allocate memory (C or C++)
void * malloc (nbytes);      // nbytes=size
char *buffer;                // declare a buffer
// allocate 10 bytes to the buffer
buffer = (char *)malloc(10);
// allocate memory (C or C++)
// nelements = number elements
// size = size of each element
void * malloc (nelements, size);
int *nums;              // declare a buffer
// allocate 5 sets of ints
nums = (char *)calloc(5,sizeof(int));
// reallocate memory (C or C++)
void * realloc (*ptr, size);
// delete memory (C or C++)
void free (*ptr);
```

ANSI C++ LIBRARY FILES

The following files are part of the ANSI C++ standard and should work in most compilers.

```
<algorithm.h>   <bitset.h>       <deque.h>      <exception.h>
<fstream.h>     <functional.h>   <iomanip.h>    <ios.h>
<iosfwd.h>      <iostream.h>     <istream.h>    <iterator.h>
<limits.h>      <list.h>         <locale.h>     <map.h>
<memory.h>      <new.h>         <numeric.h>    <ostream.h>
<queue.h>       <set.h>          <sstream.h>    <stack.h>
<stdexcept.h>   <streambuf.h>    <string.h>     <typeinfo.h>
<utility.h>     <valarray.h>     <vector.h>
```

C/C++ STANDARD LIBRARY

Only the most commonly used functions are listed. Header files without .h are in namespace std. File names are actually lower case.

STDIO.H, CSTDIO (Input/output

```
FILE* f=fopen("filename", "r"); // Open for reading,
// NULL (0) if error. Mode may also be "w" (write)
// "a" append, "a+" update, "rb" binary
fclose(f);                          // Close file f
fprintf(f, "x=%d", 3); // Print "x=3" Other conversions:
"%5d %u %-8ld" // int width 5, unsigned int, long left just.
"%o %x %X %lx" // octal, hex, HEX, long hex
"%f %5.1f"     // float or double: 123.000000, 123.0
"%e %g"        // 1.23e2, use either f or g
"%c %s"        // char, char*
"%%"           // %
sprintf(s, "x=%d", 3);  // Print to array of char s
printf("x=%d", 3);      // Print to stdout
fprintf(stderr, ...)    // Print to standard error
getc(f);       // Read one char (as an int) or EOF from f
ungetc(c, f);           // Put back one c to f
getchar();              // getc(stdin);
putc(c, f)              // fprintf(f, "%c", c);
putchar(c);             // putc(c, stdout);
fgets(s, n, f);         // Read line into char s[n] from f.
                        // NULL if EOF
gets(s)         // fgets(s, INT_MAX, no bounds check
fread(s, n, 1, f);      // Read n bytes from f to s,
                        // return number read
fwrite(s, n, 1, f);     // Write n bytes of s to f,
                        // return number written
fflush(f);              // Force buffered writes to f
fseek(f, n, SEEK_SET);// Position binary file f at n
ftell(f);               // Position in f, -1L if error
rewind(f);      // fseek(f, 0L, SEEK_SET); clearerr(f);
feof(f);                // Is f at end of file?
ferror(f);              // Error in f?
perror(s);              // Print char* s and error message
clearerr(f);            // Clear error code for f
remove("filename");     // Delete file, return 0 if OK
rename("old", "new"); // Rename file, return 0 if OK
f = tmpfile();          // Create temporary file, mode "wb+"
tmpnam(s);      // Put a unique file name in char s[L_tmpnam]
```

STDLIB.H, CSTDLIB (Misc. functions)

```
atof(s);        // Convert char* s to float,
atol(s);        // to long,
atoi(s);        // to int
rand();         // Random int 0 to RAND_MAX
srand(seed);    // reset rand()
void* p = malloc(n); // Allocate n bytes. Obsolete: use new
free(p);        // Free memory. Obsolete: use delete
exit(n);        // Kill program, return status n
system(s);      // Execute OS command s (system dependent)
getenv("PATH"); // Environment variable or 0
                // (system dependent)
abs(n); labs(ln); // Absolute value as int, long
```

STRING.H, CSTRING

Character array handling functions

Strings are type char[] with a ’\0’ in the last element used.

```
strcpy(dst, src);   // Copy string. Not bounds checked
strcat(dst, src);   // Concatenate to dst.
                    // Not bounds checked
strcmp(s1, s2);     // Compare, <0 if s1< s2,
                    //          0 if s1==s2,
                    //         >0 if s1> s2
strncpy(dst, src, n);  // Copy up to n chars,
                       // also strncat(), strncmp()
strlen(s);             // Length of s not counting \0
strchr(s,c); strrchr(s,c);  // Address of
                            // first/last char c in s or 0
strstr(s, sub);  // Address of first substring in s or 0
/* mem... functions are for any pointer types (void*),
         length n bytes */
memmove(dst, src, n);  // Copy n bytes from src to dst
memcmp(s1, s2, n);     // Compare n bytes as in strcmp
memchr(s, c, n);       // Find first byte c in s,
                       // return address or 0
memset(s, c, n);       // Set n bytes of s to c
```

CTYPE.H, CCTYPE (Character types)

```
isalnum(c);                 // Is c a letter or digit?
isalpha(c); isdigit(c);  // Is c a letter? Digit?
islower(c); isupper(c);  // Is c lower case? Upper case?
tolower(c); toupper(c);  // Convert c to lower/upper case
```

MATH.H, CMATH (Floating point math)

```
sin(x); cos(x); tan(x);      // Trig functions,
                             //  x (double) is in radians
asin(x); acos(x); atan(x);  // Inverses
atan2(y, x);                 // atan(y/x)
sinh(x); cosh(x); tanh(x);  // Hyperbolic
exp(x); log(x);              // e to the x, log base e
log10(x);                    // log base 10
pow(x, y); sqrt(x);          // x to the y, square root
ceil(x); floor(x);    //  Round up or down (as a double)
fabs(x); fmod(x, y);  // Absolute value, x mod y
```

TIME.H, CTIME (Clock)

```
clock()/CLOCKS_PER_SEC;  // Time in seconds since
                         // program started
time_t t=time(0);  // Absolute time in seconds or
                   // -1 if unknown
tm* p=gmtime(&t);  // 0 if UCT unavailable, else p->tm_X
                   // where X is:
                   // sec, min, hour, mday, mon (0-11),
                   // year (-1900), wday, yday, isdst
asctime(p);              // "Day Mon dd hh:mm:ss yyyy\n"
asctime(localtime(&t)); // Same format, local time
```

ASSERT.H, CASSERT (Debugging aid)

```
assert(e);         // If e is false, print message and abort
#define NDEBUG     // (before #include <assert.h>),
                   // turn off assert
```

NEW.H, NEW (Out of memory handler)

```
set_new_handler(handler); // Change behavior
                          // when out of memory
void handler(void) {throw bad_alloc();}  // Default
```

IOSTREAM.H, IOSTREAM (Replaces stdio.h)

```
cin >> x >> y;  // Read words x, y (any type) from stdin
cout << "x=" << 3 << endl;  // Write line to stdout
cerr << x << y << flush;    // Write to stderr and flush
c = cin.get();              // c = getchar();
cin.get(c);                 // Read char
cin.getline(s, n, '\n');    // Read line into char s[n]
                            // to '\n' (default)
if (cin)                    // Good state (not EOF)?
// To read/write any type T:
istream& operator>>(istream& i, T& x)
                            {i >> ...; x=...; return i;}
ostream& operator<<(ostream& o, const T& x)
                            {return o << ...;}
```

FSTREAM.H, FSTREAM

File I/O works like cin, cout as above

```
ifstream f1("filename");   // Open text file for reading
if (f1)                  // Test if open and input available
f1 >> x;                 // Read object from file
f1.get(s);               // Read char or line
f1.getline(s, n);        // Read line into string s[n]
ofstream f2("filename");  // Open file for writing
if (f2) f2 << x;         // Write to file
```

IOMANIP.H, IOMANIP (Output formatting)

```
cout << setw(6) << setprecision(2) << setfill('0') << 3.1;
                                          // print "003.10"
```

STRING (Variable sized character array)

```
string s1, s2="hello";  // Create strings
s1.size(), s2.size();   // Number of characters: 0, 5
s1 += s2 + ' ' + "world";  // Concatenation
s1 == "hello world"     // Comparison, also <, >, !=, etc.
s1[0];                  // 'h'
s1.substr(m, n); // Substring of size n starting at s1[m]
s1.c_str();             // Convert to const char*
getline(cin, s);        // Read line ending in '\n'
```

VECTOR

Variable sized array/stack with built in memory allocation

```
vector<int> a(10); // a[0]..a[9] int (default size is 0)
a.size();          // Number of elements (10)
a.push_back(3);    // Increase size to 11, a[10]=3
a.back()=4;        // a[10]=4;
a.pop_back();      // Decrease size by 1
a.front();         // a[0];
a[20]=1;           // Crash: not bounds checked
a.at(20)=1;        // Like a[20] but throws out_of_range()
for (vector<int>::iterator p=a.begin(); p!=a.end(); ++p)
    *p=0;          // Set all elements of a to 0
vector<int> b(a.begin(), a.end()); // b is copy of a
vector<T> c(n, x); // c[0]..c[n-1] init to x
T d[10]; vector<T> e(d, d+10); // e is initialized from d
```

DEQUE (array/stack/queue)

deque<T> is like vector<T>, but also supports:

```
a.push_front(x);   // Puts x at a[0], shifts elements
                   // toward back
a.pop_front();     // Removes a[0], shifts toward front
```

UTILITY (Pair)

```
pair<string, int> a("hello", 3);  // A 2-element struct
a.first;                           // "hello"
a.second;                          // 3
```

MAP (associative array)

```
map<string, int> a;  // Map from string to int
a["hello"]=3;          // Add or replace element a["hello"]
for (map<string, int>::iterator p=a.begin(); \
                                          p!=a.end(); ++p)
    cout << (*p).first << (*p).second; // Prints hello, 3
a.size();              // 1
```

ALGORITHM

A collection of 60 algorithms on sequences with iterators

```
min(x, y); max(x, y);  //  Smaller/larger of x, y
                       //  (any type defining <)
swap(x, y);        //  Exchange values of variables x and y
sort(a, a+n);      //  Sort array a[0]..a[n-1] by <
sort(a.begin(), a.end());  //  Sort vector or deque
```

EXAMPLES

First program in C++

p007

```
#include <iostream>
using namespace std;
int main ()
{
    cout << "Hello World!";
    return 0;
}
```

Operating with variables

p014

```
#include <iostream>
using namespace std;
int main ()
{
    // declaring variables:
    int a, b;
    int result;
    // process:
    a = 5;
    b = 2;
    a = a + 1;
    result = a - b;
    // print out the result:
    cout << result;
    // terminate the program:
    return 0;
}
```

Initialization of variables

p015

```
#include <iostream>
using namespace std;
int main ()
{
    int a=5;     // initial value = 5
    int b(2);    // initial value = 2
    int result;  // initial value undetermined
    a = a + 3;
    result = a - b;
    cout << result;
    return 0;
}
```

String

p016a

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
    string mystring = "This is a string";
    cout << mystring;
    return 0;
}
```

String

p016b

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
    string mystring;
    mystring = "This is the initial string content";
    cout << mystring << endl;
    mystring = "This is  a different string content";
    cout << mystring << endl;
    return 0;
}
```

Defined constants

p020

```
#include <iostream>
using namespace std;
#define PI 3.14159
#define NEWLINE '\n'
int main ()
{
    double r=5.0;
    double circle;
    circle = 2 * PI * r;
    cout << circle;
    cout << NEWLINE;
    return 0;
}
```

Assignment operator

p021

```
#include <iostream>
using namespace std;
int main ()
{
    int a, b;  // a:?, b:?
    a = 10;    // a:10, b:?
    b = 4;     // a:10, b:4
    a = b;     // a:4, b:4
    b = 7;     // a:4, b:7
    cout << "a:";
    cout << a;
    cout << " b:";
    cout << b;
    return 0;
}
```

Compound assignment operators

p023

```
#include <iostream>
using namespace std;
int main ()
{
    int a, b=3;
    a = b;
    a+=2;     // equivalent to a=a+2
    cout << a;
    return 0;
}
```

Conditional operator

p025

```
#include <iostream>
using namespace std;
int main ()
{
    int a,b,c;
    a=2;
    b=7;
    c = (a>b) ? a : b;
    cout << c;
    return 0;
}
```

I/O example

p031

```
#include <iostream>
using namespace std;
int main ()
{
    long int i;
    cout << "Please enter an integer value: ";
    cin >> i;
    cout << "The value you entered is " << i;
    cout << " and its double is " << i*2 << ".\n";
    return 0;
}
```

Cin with strings

p032a

```
#include <iostream>
#include <string>
using namespace std;
int main ()
{
    string mystr;
    cout << "What's your name? ";
    getline (cin, mystr);
    cout << "Hello " << mystr << ".\n";
    cout << "What is your favorite team? ";
    getline (cin, mystr);
    cout << "I like " << mystr << " too!\n";
    return 0;
}
```

Stringstreams

p032b

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
int main ()
{
    string mystr;
    float price=0;
    int quantity=0;
    cout << "Enter price: ";
    getline (cin,mystr);
    stringstream(mystr) >> price;
    cout << "Enter quantity: ";
    getline (cin,mystr);
    stringstream(mystr) >> quantity;
    cout << "Total price: " << price * quantity << endl;
    return 0;
}
```

Countdown using while

p035

```
#include <iostream>
using namespace std;
int main ()
{
    int n;
    cout << "Enter the starting number > ";
    cin >> n;
    while (n>0) {
        cout << n << ", ";
        --n;
    }
    cout << "FIRE!\n";
    return 0;
}
```

Number echoer

p036

```
#include <iostream>
using namespace std;
int main ()
{
    unsigned long n;
    do {
        cout << "Enter number (0 to end): ";
        cin >> n;
        cout << "You entered: " << n << "\n";
    } while (n != 0);
    return 0;
}
```

Countdown using a for loop

p037

```
#include <iostream>
using namespace std;
int main ()
{
    for (int n=10; n>0; n--) {
        cout << n << ", ";
    }
    cout << "FIRE!\n";
    return 0;
}
```

Break loop example

p038a

```
#include <iostream>
using namespace std;
int main ()
{
    int n;
    for (n=10; n>0; n--)
    {
        cout << n << ", ";
        if (n==3)
        {
            cout << "countdown aborted!";
            break;
        }
    }
    return 0;
}
```

Continue loop example

p038b

```
#include <iostream>
using namespace std;
int main ()
{
    for (int n=10; n>0; n--) {
        if (n==5) continue;
        cout << n << ", ";
    }
    cout << "FIRE!\n";
    return 0;
}
```

Goto loop example

p039

```
#include <iostream>
using namespace std;
int main ()
{
    int n=10;
    loop:
    cout << n << ", ";
    n--;
    if (n>0) goto loop;
    cout << "FIRE!\n";
    return 0;
}
```

Case switch

p040

```
#include <iostream>
using namespace std;
int main(){
    int x;
    cin >> x;
    switch (x) {
        case 1:
        case 2:
        case 3:
            cout << "x is 1, 2 or 3";
            break;
        default:
            cout << "x is not 1, 2 nor 3";
    }
    return 0;
}
```

Function example
p041

```
#include <iostream>
using namespace std;
int addition (int a, int b)
{
    int r;
    r=a+b;
    return (r);
}
int main ()
{
    int z;
    z = addition (5,3);
    cout << "The result is " << z;
    return 0;
}
```

Function example
p044

```
#include <iostream>
using namespace std;
int subtraction (int a, int b)
{
    int r;
    r=a-b;
    return (r);
}
int main ()
{
    int x=5, y=3, z;
    z = subtraction (7,2);
    cout << "The first result is  "
         << z << '\n';
    cout << "The second result is "
         << subtraction (7,2) << '\n';
    cout << "The third result is  "
         << subtraction (x,y) << '\n';
    z= 4 + subtraction (x,y);
    cout << "The fourth result is " << z << '\n';
    return 0;
}
```

Void function
p045

```
#include <iostream>
using namespace std;
void printmessage()
{
    cout << "I'm a function!\nI love you";
}
int main(){
    printmessage();
    return 0;
}
```

Pass parameters by reference
p047

```
#include <iostream>
using namespace std;
void duplicate (int& a, int& b, int& c)
{
    a*=2;
    b*=2;
    c*=2;
}
int main ()
{
    int x=1, y=3, z=7;
    duplicate (x, y, z);
    cout << "x=" << x << ", y=" << y << ", z=" << z;
    return 0;
}
```

More returning value
p048

```
#include <iostream>
using namespace std;
void prevnext (int x, int& prev, int& next)
{
    prev = x-1;
    next = x+1;
}
int main ()
{
    int x=100, y, z;
    prevnext (x, y, z);
    cout << "Previous=" << y << ", Next=" << z;
    return 0;
}
```

Default values in functions
p049

```
#include <iostream>
using namespace std;
int divide (int a, int b=2)
{
    int r;
    r=a/b;
    return (r);
}
int main ()
{
    cout << divide (12);
    cout << endl;
    cout << divide (20,4);
    return 0;
}
```

Overloaded function
p050

```
#include <iostream>
using namespace std;
int operate (int a, int b)
{
    return (a*b);
}
float operate (float a, float b)
{
    return (a/b);
}
int main ()
{
    int x=5,y=2;
    float n=5.0,m=2.0;
    cout << operate (x,y);
    cout << "\n";
    cout << operate (n,m);
    cout << "\n";
    return 0;
}
```

Recursion Factorial
p051

```
#include <iostream>
using namespace std;
long factorial (long a)
{
    if (a > 1)
        return (a * factorial (a-1));
    else
        return (1);
}
int main ()
{
    long number;
    cout << "Please type a number: ";
    cin >> number;
    cout << number << "! = " << factorial (number);
    return 0;
}
```

Declaring functions prototypes
p052

```
#include <iostream>
using namespace std;
void odd (int a);
void even (int a);
int main ()
{
    int i;
    do {
        cout << "Type a number (0 to exit): ";
        cin >> i;
        odd (i);
    } while (i!=0);
    return 0;
}
void odd (int a)
{
    if ((a%2)!=0) cout << "Number is odd.\n";
    else even (a);
}
void even (int a)
{
    if ((a%2)==0) cout << "Number is even.\n";
    else odd (a);
}
```

Arrays
p056

```
#include <iostream>
using namespace std;
int billy [] = {16, 2, 77, 40, 12071};
int n, result=0;
int main ()
{
    for ( n=0 ; n<5 ; n++ )
    {
        result += billy[n];
    }
    cout << result;
    return 0;
}
```

Arrays as parameters
p058

```
#include <iostream>
using namespace std;
void printarray (int arg[], int length) {
    for (int n=0; n<length; n++)
        cout << arg[n] << " ";
        cout << "\n";
}
int main ()
{
    int firstarray[]  = {5, 10, 15};
    int secondarray[] = {2, 4, 6, 8, 10};
    printarray (firstarray, 3);
    printarray (secondarray,5);
    return 0;
}
```

Null-terminated seq of chars
p062

```
#include <iostream>
using namespace std;
int main ()
{
    char question[] = "Please, enter your first name: ";
    char greeting[] = "Hello, ";
    char yourname [80];
    cout << question;
    cin  >> yourname;
    cout << greeting << yourname << "!";
    return 0;
}
```

Pointer
p066a

```
#include <iostream>
using namespace std;
int main ()
{
    int firstvalue, secondvalue;
    int * mypointer;
    mypointer = &firstvalue;
    *mypointer = 10;
    mypointer = &secondvalue;
    *mypointer = 20;
    cout << "firstvalue is  " << firstvalue  << endl;
    cout << "secondvalue is " << secondvalue << endl;
    return 0;
}
```

More pointers
p066b

```
#include <iostream>
using namespace std;
int main ()
{
    int firstvalue = 5, secondvalue = 15;
    int * p1, * p2;
    p1 = &firstvalue;  // p1 = address of firstvalue
    p2 = &secondvalue; // p2 = address of secondvalue
    *p1 = 10;    // value pointed p1 = 10
    *p2 = *p1;   // value pointed p2 = value pointed p1
    p1 = p2;     // p1 = p2 (value of pointer copied)
    *p1 = 20;    // value pointed by p1 = 20
    cout << "firstvalue is  " << firstvalue  << endl;
    cout << "secondvalue is " << secondvalue << endl;
    return 0;
}
```

More pointers
p068

```
#include <iostream>
using namespace std;
int main ()
{
    int numbers[5];
    int *p;
    p = numbers;          *p = 10;
    p++;                  *p = 20;
    p = &numbers[2];      *p = 30;
    p = numbers + 3;      *p = 40;
    p = numbers;      *(p+4) = 50;
    for (int n=0; n<5; n++)
        cout << numbers[n] << ", ";
    return 0;
}
```

Increaser
p072

```
#include <iostream>
using namespace std;
void increase (void* data, int psize) {
    if ( psize == sizeof(char) ) {
        char* pchar;
        pchar=(char*)data;
        ++(*pchar);
    } else if (psize == sizeof(int) ) {
        int* pint;
        pint=(int*)data;
        ++(*pint);
    }
}
int main () {
    char a = 'x';
    int b = 1602;
    increase (&a,sizeof(a));
    increase (&b,sizeof(b));
    cout << a << ", " << b << endl;
    return 0;
}
```

Pointer to functions
p073

```
#include <iostream>
using namespace std;
int addition (int a, int b) {
    return (a+b);
}
int subtraction (int a, int b) {
    return (a-b);
}
int operation (int x, int y,
               int (*functocall)(int,int)) {
    int g;
    g = (*functocall)(x,y);
    return (g);
}
int main () {
    int m,n;
    int (*minus)(int,int) = subtraction;
    m = operation (7, 5, addition);
    n = operation (20, m, minus);
    cout << n;
    return 0;
}
```

Dynamic memory (new)
p076

```
#include <iostream>
#include <new>
using namespace std;
int main () {
    int i,n;
    int * p;
    cout << "How many numbers would you like to type? ";
    cin >> i;
^^I// dynamic memory allocation
    p= new (nothrow) int[i];
    if (p == 0)
        cout << "Error: memory could not be allocated";
    else {
        for (n=0; n<i; n++) {
            cout << "Enter number: ";
            cin >> p[n];
        }
        cout << "You have entered: ";
        for (n=0; n<i; n++)
            cout << p[n] << ", ";
        /// free memory after use
        delete[] p;
    }
    return 0;
}
```

Structures
p078

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
struct movies_t {
    string title;
    int year;
} mine, yours;
void printmovie (movies_t movie);
int main () {
    string mystr;
    mine.title = "2001 A Space Odyssey";
    mine.year = 1968;
    cout << "Enter title: ";
    getline (cin,yours.title);
    cout << "Enter year: ";
    getline (cin,mystr);
    stringstream(mystr) >> yours.year;
    cout << "My favorite movie is:\n ";
    printmovie (mine);
    cout << "And yours is:\n ";
    printmovie (yours);
    return 0;
}
void printmovie (movies_t movie) {
    cout << movie.title;
    cout << " (" << movie.year << ")\n";
}
```

Array of structures

p079

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
#define N_MOVIES 3
struct movies_t {
    string title;
    int year;
} films [N_MOVIES];
void printmovie (movies_t movie);
int main () {
    string mystr;
    int n;
    for (n=0; n<N_MOVIES; n++) {
        cout << "Enter title: ";
        getline (cin,films[n].title);
        cout << "Enter year: ";
        getline (cin,mystr);
        //          string -> int
        stringstream(mystr) >> films[n].year;
    }
    cout << "\nYou have entered these movies:\n";
    for (n=0; n<N_MOVIES; n++)
        printmovie (films[n]);
    return 0;
}
void printmovie (movies_t movie) {
    cout << movie.title;
    cout << " (" << movie.year << ")\n";
}
```

Pointers to structures

p080

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;
struct movies_t {
    string title;
    int year;
};
int main () {
    string mystr;
    movies_t amovie;
    movies_t * pmovie;
    pmovie = &amovie;
    cout << "Enter title: ";
    getline (cin, pmovie->title);
    cout << "Enter year: ";
    getline (cin, mystr);
    (stringstream) mystr >> pmovie->year;
    cout << "\nYou have entered:\n";
    cout << pmovie->title;
    cout << " (" << pmovie->year << ")\n";
    return 0;
}
```

Classes example

p087

```
#include <iostream>
using namespace std;
class CRectangle {
        int x, y;
    public:
        void set_values (int,int);
        int area () {
            return (x*y);
        }
};
void CRectangle::set_values (int a, int b) {
    x = a;
    y = b;
}
int main () {
    CRectangle rect;
    rect.set_values (3,4);
    cout << "area: " << rect.area();
    return 0;
}
```

One class, two objects

p088

```
#include <iostream>
using namespace std;
class CRectangle {
        int x, y;
    public:
        void set_values (int,int);
        int area () {
            return (x*y);
        }
};
void CRectangle::set_values (int a, int b) {
    x = a;
    y = b;
}
int main () {
    CRectangle rect, rectb;
    rect.set_values (3,4);
    rectb.set_values (5,6);
    cout << "rect area:  " << rect.area()  << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

Class constructor

p089

```
#include <iostream>
using namespace std;
class CRectangle {
        int width, height;
    public:
        CRectangle (int,int);
        int area () {
            return (width*height);
        }
};
CRectangle::CRectangle (int a, int b) {
    width = a;
    height = b;
}
int main () {
    CRectangle rect (3,4);
    CRectangle rectb (5,6);
    cout << "rect area:  " << rect.area()  << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

Constructors and destructors

p090

```
#include <iostream>
using namespace std;
class CRectangle {
        int *width, *height;
    public:
        CRectangle (int,int);
        ~CRectangle ();
        int area () {
            return (*width * *height);
        }
};
CRectangle::CRectangle (int a, int b) {
    width   = new int;
    height  = new int;
    *width  = a;
    *height = b;
}
CRectangle::~CRectangle () {
    delete width;
    delete height;
}
int main () {
    CRectangle rect (3,4), rectb (5,6);
    cout << "rect area:  " << rect.area()  << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

Overloading class constructors

p091

```
#include <iostream>
using namespace std;
class CRectangle {
        int width, height;
    public:
        CRectangle ();
        CRectangle (int,int);
        int area (void) {
            return (width*height);
        }
};
CRectangle::CRectangle () {
    width  = 5;
    height = 5;
}
CRectangle::CRectangle (int a, int b) {
    width  = a;
    height = b;
}
int main () {
    CRectangle rect (3,4);
    CRectangle rectb;
    cout << "rect area:  " << rect.area()  << endl;
    cout << "rectb area: " << rectb.area() << endl;
    return 0;
}
```

Pointer to classes

p093

```
#include <iostream>
using namespace std;
class CRectangle {
        int width, height;
    public:
        void set_values (int, int);
        int area (void) {
            return (width * height);
    }
};
void CRectangle::set_values (int a, int b) {
    width = a;
    height = b;
}
int main () {
    CRectangle a, *b, *c;
    CRectangle * d = new CRectangle[2];
    b= new CRectangle;
    c= &a;
    a.set_values (1,2);
    b->set_values (3,4);
    d->set_values (5,6);
    d[1].set_values (7,8);
    cout << "a area: " << a.area() << endl;
    cout << "*b area: " << b->area() << endl;
    cout << "*c area: " << c->area() << endl;
    cout << "d[0] area: " << d[0].area() << endl;
    cout << "d[1] area: " << d[1].area() << endl;
    delete[] d;
    delete b;
    return 0;
}
```

Vectors: overloading operators

p096

```
#include <iostream>
using namespace std;
class CVector {
    public:
        int x,y;
        CVector () {};
        CVector (int,int);
        CVector operator + (CVector);
};
CVector::CVector (int a, int b) {
    x = a;
    y = b;
}
CVector CVector::operator+ (CVector param) {
    CVector temp;
    temp.x = x + param.x;
    temp.y = y + param.y;
    return (temp);
}
int main () {
    CVector a (3,1);
    CVector b (1,2);
    CVector c;
    c = a + b;
    cout << c.x << "," << c.y;
    return 0;
}
```

This

p098

```
#include <iostream>
using namespace std;
class CDummy {
    public:
        int isitme (CDummy& param);
};
int CDummy::isitme (CDummy& param) {
    if (&param == this) return true;
    else return false;
}
int main () {
    CDummy a;
    CDummy* b = &a;
    if ( b->isitme(a) )
        cout << "yes, &a is b";
    return 0;
}
```

Static members in classes

p099

```
#include <iostream>
using namespace std;
class CDummy {
    public:
        static int n;
        CDummy () { n++; };
        ~CDummy () { n--; };
};
int CDummy::n=0;
int main () {
    CDummy a;                        //  1
    CDummy b[5];                     // +5
    CDummy * c = new CDummy;        // +1
    cout << a.n << endl;            // >>7
    delete c;                       // -1
    cout << CDummy::n << endl;      // >>6
    return 0;
}
```

Friend functions

p100

```
#include <iostream>
using namespace std;
class CRectangle {
        int width, height;
    public:
        void set_values (int, int);
        int area () {
            return (width * height);           // 4*6=24
        }
        friend CRectangle duplicate (CRectangle);
};
void CRectangle::set_values (int a, int b) {
    width = a;
    height = b;
}
CRectangle duplicate (CRectangle rectparam) {
    CRectangle rectres;
    rectres.width = rectparam.width*2;         // 2*2=4
    rectres.height = rectparam.height*2;       // 2*3=6
    return (rectres);
}
int main () {
    CRectangle rect, rectb;
    rect.set_values (2,3);                     // << 2,3
    rectb = duplicate (rect);                  //    24
    cout << rectb.area();                      // >> 24
    return 0;
}
```

Friend class

p101

```
#include <iostream>
using namespace std;
class CSquare;
class CRectangle {
        int width, height;
    public:
        int area () {
            return (width * height);
        }
//          void convert (CSquare a);
        void convert (CSquare);
};
class CSquare {
    private:
        int side;
    public:
        void set_side (int a) {
            side=a;
        }
        friend class CRectangle;
};
void CRectangle::convert (CSquare a) {
    width = a.side;
    height = a.side;
}
int main () {
    CSquare sqr;
    CRectangle rect;
    sqr.set_side(4);
    cout << sqr.side << endl;
    rect.convert(sqr);
    cout << rect.area();
    return 0;
}
```

Derived classes

p103

```
#include <iostream>
using namespace std;
class CPolygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b) {
            width=a;
            height=b;
        }
};
class CRectangle: public CPolygon {
    public:
        int area () {
            return (width * height);
        }
};
class CTriangle: public CPolygon {
    public:
        int area () {
            return (width * height / 2);
        }
};
int main () {
    CRectangle rect;
    CTriangle trgl;
    rect.set_values (4,5);
    trgl.set_values (4,5);
    cout << rect.area() << endl;
    cout << trgl.area() << endl;
    return 0;
}
```

Constructors and derived classes

p105

```
#include <iostream>
using namespace std;
class mother {
    public:
        mother () {
            cout << "mother: no parameters\n";
        }
        mother (int a) {
            cout << "mother: int parameter\n";
        }
};
class daughter : public mother {
    public:
        // nothing specified: call default mother
        daughter (int a) {
            cout << "daughter: int parameter\n\n";
        }
};
class son : public mother {
    public:
        // constructor mother specified: call mother(int)
        son (int a) : mother (a) {
            cout << "son: int parameter\n\n";
        }
};
int main () {
    daughter cynthia (0);
    son daniel(0);
    return 0;
}
```

Multiple inheritance

p106

```
#include <iostream>
using namespace std;
class CPolygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b) {
            width=a;
            height=b;
        }
};
class COutput {
    public:
        void output (int i);
};

void COutput::output (int i) {
    cout << i << endl;
}
class CRectangle: public CPolygon, public COutput {
    public:
        int area () {
            return (width * height);
        }
};
class CTriangle: public CPolygon, public COutput {
    public:
        int area () {
            return (width * height / 2);
        }
};
int main () {
    CRectangle rect;
    CTriangle trgl;
    rect.set_values (4,5);
    trgl.set_values (4,5);
    rect.output (rect.area());
    trgl.output (trgl.area());
    return 0;
}
```

Pointers to base class

p107

```
#include <iostream>
using namespace std;
class CPolygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b) {
            width=a;
            height=b;
        }
};
class CRectangle: public CPolygon {
    public:
        int area () {
            return (width * height);
        }
};
class CTriangle: public CPolygon {
    public:
        int area () {
            return (width * height / 2);
        }
};
int main () {
    CRectangle rect;
    CTriangle trgl;
    CPolygon * ppoly1 = &rect;
    CPolygon * ppoly2 = &trgl;
    ppoly1->set_values (4,5);
    ppoly2->set_values (4,5);
    cout << rect.area() << endl;
    cout << trgl.area() << endl;
    return 0;
}
```

Virtual members

p108

```
#include <iostream>
using namespace std;
class CPolygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b) {
            width=a;
            height=b;
        }
        virtual int area () {
            return (0);
        }
};
class CRectangle: public CPolygon {
    public:
        int area () {
            return (width * height);
        }
};
class CTriangle: public CPolygon {
    public:
        int area () {
            return (width * height / 2);
        }
};
int main () {
    CRectangle rect;
    CTriangle trgl;
    CPolygon poly;
    CPolygon * ppoly1 = &rect;
    CPolygon * ppoly2 = &trgl;
    CPolygon * ppoly3 = &poly;
    ppoly1->set_values (4,5);
    ppoly2->set_values (4,5);
    ppoly3->set_values (4,5);
    cout << ppoly1->area() << endl;
    cout << ppoly2->area() << endl;
    cout << ppoly3->area() << endl;
    return 0;
}
```

Abstract base class

p110

```
#include <iostream>
using namespace std;
class CPolygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b) {
            width=a;
            height=b;
        }
        virtual int area (void) =0;
};
class CRectangle: public CPolygon {
    public:
        int area (void) {
            return (width * height);
        }
};
class CTriangle: public CPolygon {
    public:
        int area (void) {
            return (width * height / 2);
        }
};
int main () {
    CRectangle rect;
    CTriangle trgl;
    CPolygon * ppoly1 = &rect;
    CPolygon * ppoly2 = &trgl;
    ppoly1->set_values (4,5);
    ppoly2->set_values (4,5);
    cout << ppoly1->area() << endl;
    cout << ppoly2->area() << endl;
//    cout << ppoly1.area() << endl;
//    cout << ppoly2.area() << endl;
    return 0;
}
```

Virtual members class call

p111

```
// pure virtual members can be called
// from the abstract base class
#include <iostream>
using namespace std;
class CPolygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b) {
            width=a;
            height=b;
        }
        virtual int area (void) =0;
        void printarea (void) {
            cout << this->area() << endl;
        }
};
class CRectangle: public CPolygon {
    public:
        int area (void) {
            return (width * height);
        }
};
class CTriangle: public CPolygon {
    public:
        int area (void) {
            return (width * height / 2);
        }
};
int main () {
    CRectangle rect;
    CTriangle trgl;
    CPolygon * ppoly1 = &rect;
    CPolygon * ppoly2 = &trgl;
    ppoly1->set_values (4,5);
    ppoly2->set_values (4,5);
    ppoly1->printarea();
    ppoly2->printarea();
    return 0;
}
```

Dynamic allocation/polymorphism

p112

```
#include <iostream>
using namespace std;
class CPolygon {
    protected:
        int width, height;
    public:
        void set_values (int a, int b) {
            width=a;
            height=b;
        }
        virtual int area (void) =0;
        void printarea (void) {
            cout << this->area() << endl;
        }
};
class CRectangle: public CPolygon {
    public:
        int area (void) {
            return (width * height);
        }
};
class CTriangle: public CPolygon {
    public:
        int area (void) {
            return (width * height / 2);
        }
};
int main () {
    CPolygon * ppoly1 = new CRectangle;
    CPolygon * ppoly2 = new CTriangle;
    ppoly1->set_values (4,5);
    ppoly2->set_values (4,5);
    ppoly1->printarea();
    ppoly2->printarea();
    delete ppoly1;
    delete ppoly2;
    return 0;
}
```

Function template

p114

```
#include <iostream>
using namespace std;
template <class T> T GetMax (T a, T b) {
    T result;  /* result will be an object of the same
                  type as the parameters a and b when
                  the function template is instantiated
                  with a specific type. */
    result = (a>b)? a : b;
    return (result);
}
int main () {
    int i=5, j=6, k;
    long l=10, m=5, n;
    k=GetMax<int>(i,j);
    n=GetMax<long>(l,m);
    cout << k << endl;
    cout << n << endl;
    return 0;
}
```

Function template II

p115

```
#include <iostream>
using namespace std;
template <class T>
T GetMax (T a, T b) {
    return (a>b?a:b);
}
int main () {
    int i=5, j=6, k;
    long l=10, m=5, n;
    k=GetMax(i,j);
    n=GetMax(l,m);
    cout << k << endl;
    cout << n << endl;
    return 0;
}
```

Class templates

p116

```
#include <iostream>
using namespace std;
template <class T>
class mypair {
        T a, b;
    public:
        mypair (T first, T second) {
            a=first;
            b=second;
        }
        T getmax ();
};
template <class T>       //  T : template parameter
T mypair<T>::getmax () { //  T1 : function return type
                         //  T2 : function template param
    T retval;
    retval = a>b? a : b;
    return retval;
}
int main () {
    mypair <int> myobject (100, 75);
    cout << myobject.getmax() << endl;
    mypair <float> myobjectf (10.0, 75.5);
    cout << myobjectf.getmax() << endl;
    return 0;
}
```

Template specialization

p117

```
#include <iostream>
using namespace std;
// class template:
template <class T>
class mycontainer {
        T element;
    public:
        mycontainer (T arg) {
            element=arg;
        }
        T increase () {
            return ++element;
        }
};
// class template specialization:
template <>
class mycontainer <char> {
        char element;
    public:
        mycontainer (char arg) {
            element=arg;
        }
        char uppercase () {
            if ((element>='a')&&(element<='z'))
                element+='A'-'a';
            return element;
        }
};
int main () {
    mycontainer<int> myint (7);
    mycontainer<char> mychar ('q');
    cout << myint.increase() << endl;
    cout << mychar.uppercase() << endl;
    mycontainer<float> myfloat (5.125);
    cout << myfloat.increase() << endl;
    return 0;
}
```

Sequence template

p118

```
#include <iostream>
using namespace std;
template <class T, int N>
class mysequence {
        T memblock [N];
    public:
        void setmember (int x, T value);
        T getmember (int x);
};
template <class T, int N>
void mysequence<T,N>::setmember (int x, T value) {
    memblock[x]=value;
}
template <class T, int N>
T mysequence<T,N>::getmember (int x) {
    return memblock[x];
}
int main () {
    mysequence <int,5> myints;
    mysequence <double,5> myfloats;
    myints.setmember (0,100);
    myfloats.setmember (3,3.1416);
    cout << myints.getmember(0) << '\n';
    cout << myfloats.getmember(3) << '\n';
    cout << myfloats.getmember(4) << '\n';
    mysequence <char,5> mychars;
    mychars.setmember(1,'A');
    mychars.setmember(2,'z');
    cout << mychars.getmember(1) << '\n';
    cout << mychars.getmember(2) << '\n';
    cout << mychars.getmember(3) << '\n';
    return 0;
}
```

Namespaces

p120

```
#include <iostream>
using namespace std;
namespace first {
    int var = 5;
}
namespace second {
    double var = 3.1416;
}
int main () {
    cout << first::var << endl;
    cout << second::var << endl;
    return 0;
}
```

Using

p121a

```
#include <iostream>
using namespace std;
namespace first {
    int x = 5;
    int y = 10;
}
namespace second {
    double x = 3.1416;
    double y = 2.7183;
}
int main () {
    using first::x;
    using second::y;
    cout << x << endl;
    cout << y << endl;
    cout << first::y << endl;
    cout << second::x << endl;
    return 0;
}
```

Using

p121b

```
#include <iostream>
using namespace std;
namespace first {
    int x = 5;
    int y = 10;
}
namespace second {
    double x = 3.1416;
    double y = 2.7183;
}
int main () {
    using namespace first;
    cout << x << endl;
    cout << y << endl;
    cout << second::x << endl;
    cout << second::y << endl;
    return 0;
}
```

Using namespace example

p122

```
#include <iostream>
using namespace std;
namespace first {
    int x = 5;
}
namespace second {
    double x = 3.1416;
}
int main () {
    {
        using namespace first;
        cout << x << endl;
    }
    {
        using namespace second;
        cout << x << endl;
    }
    return 0;
}
```

Exceptions

p123

```
#include <iostream>
using namespace std;
int main () {
    try {
        throw 20;
    }
    catch (int e) {
        cout << "An exception occurred. ";
        cout << "Exception Nr. " << e << endl;
    }
    return 0;
}
```

Standard exceptions

p125

```
#include <iostream>
#include <exception>
using namespace std;
class myexception: public exception {
    virtual const char* what() const throw() {
        return "My exception happened";
    }
} myex;
int main () {
    try {
        throw myex;
    } catch (exception& e) {
        cout << e.what() << endl;
    }
    return 0;
}
```

Bad_alloc standard exception

p126

```
#include <iostream>
#include <exception>
using namespace std;
int main () {
    try {
//          int* myarray= new int[1000];
        float* myarray= new float[1000000000];
    } catch (exception& e) {
        cout << "Standard exception: " << e.what() << endl;
    }
    return 0;
}
```

Class type-casting

p128

```
#include <iostream>
using namespace std;
class CDummy {
    float i,j;
    public:
        void output () {
        cout << i <<endl;
        cout << j <<endl;
        }
};
class CAddition {
        int x,y;
    public:
        CAddition (int a, int b) {
            x=a;
            y=b;
        }
        int result() {
            return x+y;
        }
};
int main () {
    CDummy d;
    CAddition * padd;
    padd = (CAddition*) &d;
    cout << padd->result() << endl;
    d.output() ;
    return 0;
}
```

Dynamic_cast

p129

```
#include <iostream>
#include <exception>
using namespace std;
class CBase {
    virtual void dummy() {}
};
class CDerived: public CBase {
    int a;
};
int main () {
    try {
        CBase * pba = new CDerived;
        CBase * pbb = new CBase;
        CDerived * pd;
        pd = dynamic_cast<CDerived*>(pba);
        if (pd==0) cout << "Null pointer on first type-cast"
                        << endl;
        pd = dynamic_cast<CDerived*>(pbb);
        if (pd==0) cout << "Null pointer on second type-cast"
                        << endl;
    } catch (exception& e) {
        cout << "Exception: " << e.what();
    }
    return 0;
}
```

Const_cast

p131a

```
#include <iostream>
using namespace std;
void print (char * str) {
    cout << str << endl;
}
int main () {
    const char * c = "sample text";
    print ( const_cast<char *> (c) );
    return 0;
}
```

Typeid

p131b

```
#include <iostream>
#include <typeinfo>
using namespace std;
int main () {
    int * a,b;
    a=0;
    b=0;
    if (typeid(a) != typeid(b)) {
        cout << "a and b are of different types:\n";
        cout << "a is: " << typeid(a).name() << '\n';
        cout << "b is: " << typeid(b).name() << '\n';
    }
    return 0;
}
```

Typeid, polymorphic class

p132

```
#include <iostream>
#include <typeinfo>
#include <exception>
using namespace std;
class CBase {
    virtual void f() {}
};
class CDerived : public CBase {};
int main () {
    try {
        CBase* a = new CBase;
        CBase* b = new CDerived;
        cout << "a is: " << typeid(a).name() << '\n';
        cout << "b is: " << typeid(b).name() << '\n';
        cout << "*a is: " << typeid(*a).name() << '\n';
        cout << "*b is: " << typeid(*b).name() << '\n';
    } catch (exception& e) {
        cout << "Exception: " << e.what() << endl;
    }
    return 0;
}
```

Function macro

p133

```
#include <iostream>
using namespace std;
#define getmax(a,b) ((a)>(b)?(a):(b))
int main() {
    int x=5, y;
    y= getmax(x,2);
    cout << y << endl;
    cout << getmax(7,x) << endl;
    return 0;
}
```

Standard macro names

p137

```
#include <iostream>
using namespace std;
int main() {
    cout << "This is the line number " << __LINE__;
    cout << " of file " << __FILE__ << ".\n";
    cout << "Its compilation began " << __DATE__;
    cout << " at " << __TIME__ << ".\n";
    cout << "The compiler gives a __cplusplus value of ";
    cout << __cplusplus;
    return 0;
}
```

Basic file operations

p138

```
#include <iostream>
#include <fstream>
using namespace std;
int main () {
    ofstream myfile;
    myfile.open ("example.txt");
    myfile << "Writing this to a file.\n";
    myfile.close();
    return 0;
}
```

Writing on a text/bin file

p140

```
#include <iostream>
#include <fstream>
using namespace std;
int main () {
//    ofstream myfile ("example.txt");
    ofstream myfile ("example.bin",
                          ios::out | ios::app | ios::binary);

    if (myfile.is_open()) {
        myfile << "This is a line.\n";
        myfile << "This is another line.\n";
        myfile.close();
    } else cout << "Unable to open file";
    return 0;
}
```

Reading a text file

p141

```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;
```

```
int main () {
    string line;
    ifstream myfile ("example.txt");
    if (myfile.is_open()) {
        while (! myfile.eof() ) {
            getline (myfile,line);
            cout << line << endl;
        }
        myfile.close();
    } else cout << "Unable to open file";
    return 0;
}
```

Obtaining file size

p143a

```
#include <iostream>
#include <fstream>
using namespace std;
int main () {
    long begin,end;
    ifstream myfile ("example.txt");
    begin = myfile.tellg();
    myfile.seekg (0, ios::end);
    end = myfile.tellg();
    myfile.close();
    cout << "size is: " << (end-begin) << " bytes.\n";
    return 0;
}
```

Reading a complete binary file

p143b

```
#include <iostream>
#include <fstream>
using namespace std;
ifstream::pos_type size;
char * memblock;
int main () {
    ifstream file ("example.bin",
                   ios::in|ios::binary|ios::ate);
    if (file.is_open()) {
        size = file.tellg();
        memblock = new char [size];
        file.seekg (0, ios::beg);
        file.read (memblock, size);
        file.close();
        cout << "the complete file content is in memory";
        delete[] memblock;
    } else cout << "Unable to open file";
    return 0;
}
```

Based on:

C++ Reference Card, 2002, The Book Company Storrs, CT

The c++ Language Tutorial, 2007, Juan Soulie

Θανάσης Νάτσης 8/5/2016