BASICS : C versus C++, program form, data types, strings, constants ...

C VERSUS C++

C is a subset of C++. A well-written C program file should compile on a C++ compiler with little or no changes. If your wondering whether you should learn one or the other - don't. Learn C++ and you'll get the benefits of both C and C++.

As a note, every program in this reference has been compiled and run to verify syntax and assure that the code snippets are correct.

PROGRAM FORM

As an ongoing tradition, the first program you write should be the infamous "Hello World" program.

```
// hello.cp
#include <iostream.h>

void main( void )

   cout << "hello world" << endl;

// end hello.cp
```

To be truely portable, a C/C++ program should declare 'main' to return an integer value. If a program is successful, it should return a zero. The following program is identical to the previous one except it uses the more standard protocol.

```
// portable.cp
#include <iostream.h>

int main( void )        // portable C code uses 'int main'

   cout << "hello world" << endl;

   return 0;               // successful program termination

// end portable.cp
```

Returning a value of '1' generally represents an unsuccessful program termination. You can also use the EXIT_SUCCESS and/or EXIT_FAILURE definitions located in the standard library file (i.e., stdlib.h).□□ Most of the sample programs in this reference use the clearer, but slightly less portable 'void main' declaration.

Anything on a line after a double slash (//) is ignored by the compiler and is the standard way of commenting code in C++. Another way to comment your code is to use the C-style slash-asterisk method. The following program "square.cp" is almost as simple as the 'hello world' program, except it demonstrates the basics of using functions as well as both styles of comments.

```
// square.cp
/*
    Sometimes it convenient to use the "old" C-style comments,
    especially when you have more than one line of stuff to say.
*/

#include <iostream.h> // standard library.

#define MAX 10            // preprocessor directive.
```

```
int DoSquare(int);       // function prototypes required in C++.

void main( void )        // main is the start of all C programs.

   int n;
   for(n=1;n<MAX;n++)
      cout << n << " squared = " << DoSquare(n) << endl;


int DoSquare(int m)     // function definition.

   m = m * m;
   return m;                // return value.

// end square.cp
```

White space is ignored in C/C++. Brackets are used to group (or enclose) multiple statements. Function calls are depicted with parenthesis () at the end of a function name which enclose variables which are used during the function call. Semicolons depict the end of a statement (i.e., you can place multiple statements on a single line separated by semicolons if you wanted).

DATA TYPES AND TYPE CONVERSION

The following program identifies the built-in data types supported by C and methods for converting between these data types.

```
// types.cp
#include <iostream.h>

void main( void )

   // declarations
   char                 c = 'A';   // usually 1-byte long.
   short int           si= 1;      // minimum range +/-32767.
   short                s = 2;      // short same as short int.
   int                  i = 3;      // minimum range +/-32767.
   long int            li= 4;      // minimum range +/-2147483647.
   long                 l = 5;      // long same as long int.
   float                f = 10.1; // min 6 digits (decimal) precision.
   double               d = 11.2; // min 10 digits (decimal) precision.
   long double          ld= 12.3;

   unsigned char        uc;         // unsigned integers can only store
   unsigned short int   usi;        // positive numbers.
   unsigned int         ui;
   unsigned long int    uli;

   signed char          sc;         // signed integers can store positive
   signed short int     ssi;        // or negative numbers.
   signed int           si2;
   signed long int      sli;

   // simple assignment
   s = 3;
   cout << s << endl;

   // automatic conversion
   f = s * i / d;
   cout << f << endl;
```

```
    // inline conversions
    ld = (long double)i;
    s = short(d);
    cout << ld << endl << s << endl;

  // end types.cp
```

STRINGS

Strings in C are "null terminated". This means that the following declaration;

```
  char *str = "Test string";
```

allocates a block of memory 12 bytes long and returns a pointer 'str' to the first character in the string. The first 11 bytes contain the characters "Test string", the 12th byte contains NULL (i.e., \0 or zero). To declare an empty string, use the following:

```
  char *nullStr = "";
```

Since the MacOS is based on Pascal strings, it is important that the difference be discussed. The same string in Pascal would also be 12 bytes long, however, the first character in the string is actually the integer '11' which defines how many characters are in the string. Your compiler should provide functions such as CtoPstr() and PtoCstr() to convert between the two formats. If you want to declare a Pascal-style string in C, use the following:

```
  char *Pstr = "\pThis is a Pascal string";
```

CONSTANTS

```
  // constant definition examples
  #define INTEGER             123
  #define LONG                123L
  #define UNSIGNED_LONG     123UL
  #define FLOAT              12.3F
  #define CHAR_CONST          'A'
  #define OCTAL              037              /* leading zero */
  #define HEXADECIMAL         0X5
  #define STRING             "characters"

  // escape sequences
  #define ALERT              \a
  #define BACKSPACE           \b
  #define FORMFEED            \f
  #define NEWLINE            \n
  #define CARRIAGE_RETURN \r
  #define HORIZ_TAB           \t
  #define VERT_TAB            \v
  #define BACKSLASH           \\
  #define QUESTION_MARK     \?
  #define SINGLE_QUOTE       \'
  #define DOUBLE_QUOTE       \"
  #define OCTAL_SEQ           \o13             /* vertical tab */
  #define HEX_SEQ            \x7              /* alert */

  // enumeration constants
  enum boolean  FALSE, TRUE ;
  enum identifiers  first = 1, second, third ;
```

The first item in an enumeration statement has a value of 0, the next 1, the next 2, etc. unless specified values are

provided.

TERMINOLOGY

There's a lot of fancy terminology in C++ which tends to confuse the novice. Here are some of the buzz words that are used in object-oriented programming and what they mean:

Instances - Variables of a pre-defined class type are called instances of that class, or objects.

Objects - Objects are variables which are instances of a class. Here is a simple example:

```
class Student               // define class Student

   char *name;
   char *address;
   int   id;

   void input( void );   // class methods
   void print( void );


Student freshman;          // freshman is an object,
                           // a class variable, and
                           // an instance of the class Student.
```

Methods - Methods are nothing more than member functions of a class.

Polymorphism - When the same function works on different types of objects, it's called polymorphism. For example, you OPEN a door, you OPEN a window, and you OPEN your eyes. Polymorphism means "many different forms".

Encapsulation - The combination of member data and member functions (or methods) into an object. With C++, you assign an object data attributes and methods which operate on the data. Generally speaking, all of the code for a particular class of objects is maintained in a separate source code file, or in other words, it's encapsulated.

Inheritance - One of the coolest things about C++ is that you can create an object which inherits all of the capabilities of a parent object, and then make minor modifications to suit your needs. For example, if there was a window class which did everything you needed except for, say, window resizing. You could create a subclass of the window class and then add the code needed to perform the task. There are two important things to realize: 1) you only need to add the code to perform the specific task you need, and; 2) you don't need the original source code of the parent class to make modifications.

Multiple Inheritance - Inheriting capabilities or traits from more than one parent defines multiple inheritance. There are some problems with using multiple inheritance and many programmers avoid it at all costs (similar to using goto statements), but there are times when it can be useful.

Virtual Function (or method) - Allows polymorphism by defining a function which is called by an object at runtime (i.e., late binding) instead of at compile time (i.e., early binding).