

C++

SYNOPSIS

This document describes the various implementations of the C++ Language running under various implementations of the UNIX operating system. Topics include creation, compilation, and execution of a C++ program. Also covered in a separate section is a discussion of user-created C++ object libraries. It is assumed that the reader has a working knowledge of C++ and a UNIX text editor, such as vi.

C++ LANGUAGE INTRODUCTION

There are many implementations of the C++ language available on the various UNIX systems at Cal Poly. Within this document we will be addressing the compilers which fall into the C++ category as opposed to "standard" C or Objective C which are addressed in their own respective sections.

The compilers involved boil down to either the system supplied compiler "CC" (and some of its variants) and the GNU Project C compiler named "g++". The following table describes the platform, the name of the compiler, and any special considerations.

Hardware Platform	Command Name	Compiler Vendor
RISC/6000 AIX 3	<i>xlc, ixlc, icc, or xlc_r</i>	IBM C Set ++ Compiler
Any Platform	<i>gcc</i>	GNU Project C++ Compiler

NOTE: Where multiple commands exist for a compiler, consult the man pages for that command to determine which options are defaulted with each command variation.

NAMING CONVENTIONS

C++ source programs must follow standard UNIX naming conventions. The C source file must end with ". C" for most implementations. The GNU Project C++ Compiler accepts additional extensions which include ". C", ". cc", ". cxx", and several others. Please refer to the man page for the specific compiler parameters and possible extensions.

CREATE A SOURCE FILE

Use any available text editor, such as vi, to create your source file. (See the Information Systems User Guide "vi Editor" which is included in the "Introduction to UNIX" User Guide bundle from El Corral Bookstore.) You may also bring the source in from another system using such facilities as Kermit, FTP, or xmodem.

COMPILE THE C SOURCE PROGRAM

Most of the C compilers have many flags in common as well as the syntax of the command. In the following example, replace the word "cc_cmd" with the name of the compiler from the table on page 1. The general syntax of C compilers is

```
% cc_cmd [-c] [-Dname[=value]] [-E] [-g] [-Ipathname] [-lkey]
          [-Lpathname] [-o executable_name] [-O] [-Uname] [-w] sourcefile
          [sourcefile ... sourcefile]<CR>
```

where the options are defined as

-c

Compile the source only, do not call the linker (ld).

- <i>Dname</i> [=value]	Define <i>name</i> as in a #define directive. If =value is not specified, 1 is assumed.
- <i>E</i>	Preprocess, but do not compile; output to stdout (standard output).
- <i>g</i>	Produce debug information for use with debuggers such as "dbx".
- <i>Ipathname</i>	Search the directory specified by " <i>pathname</i> " for include files which do not start with an absolute path.
- <i>Ikey</i>	Selects the library libkey.a to be searched for unresolved references by <i>ld</i> .
- <i>Lpathname</i>	Search the directory specified by " <i>pathname</i> " for the libraries specified by <i>-Ikey</i> .
- <i>o name</i>	Name the executable file <i>name</i> instead of a.out.
- <i>O</i>	Optimize code generation.
- <i>Uname</i>	Undefine name as in #undef directive.
- <i>w</i>	Suppress informational, language-level, and warning messages.
<i>sourcefile</i> [<i>sourcefile</i> ... <i>sourcefile</i>]	one or more source filenames.

Additional parameters may be found by viewing the man page for the compiler on its system by entering

```
% man cc_cmd<CR>
```

where "cc_cmd" is the name of the compiler command you wish to use.

EXECUTE A C++ EXECUTABLE

To execute your compiled program, you simply type the name of the object module which was produced from the compilation step. The default for both compilers is "a.out". CC does have the capability of producing an object module with a different name, but for the purposes of this example, we will assume that the object module name is "a.out". To execute it you enter

```
% a.out<CR> This will cause the program to be loaded and begin execution.
```

COMPILING IN MULTIPLE STEPS

Let us assume in this example we have a program that is composed of three different source files. These source files ("myprog1.C", "myprog2.C", and "myprog3.C") are all located in the current working directory. To compile them all, then produce a single object file, would be accomplished by executing the CC or the ld commands. This is done by typing

```
% CC myprog1.C myprog2.C myprog3.C<CR>
```

This can also be accomplished in multiple steps by entering

```
% CC -c myprog1.C<CR>
% CC -c myprog2.C<CR>
```

```
% CC -c myprog3. C<CR>
% ld myprog1. o myprog2. o myprog3. o -lc<CR>
```

or

```
% CC -c myprog1. C<CR>
% CC -c myprog2. C<CR>
% CC myprog3. C myprog1. o myprog2. o<CR>
```

This compiles the three separate source files. They are then linked to the standard C library to produce an object module named "a. out". **NOTE:** When "ld" is used to link the modules together, the C library must be specified and placed in order. For example

```
% ld myprog3. o myprog1. o myprog2. o -lc<CR>
```

To compile and link the routine "myprog. C" to additional system subroutine library files "libcurs. a" and "libcurses. a", you would type

```
% CC myprog. C -lcurs -lcurses<CR>
```

Note that only the portion "curs" and "curses" are used from the library file name. When this is done as part of the "ld" step or as part of the CC command, the additional subroutine libraries must be indicated as the last items on the command line.

HOW TO USE THE INTERACTIVE DEBUGGER

There is an interactive debugger available for each of the C++ compilers on most of the campus UNIX systems. Please refer to the section entitled "Debugging Tools" for more information on using these interactive debuggers.

COMMAND SUMMARY

```
% CC filename<CR>
```

Compile the C++ source file specified by *filename* and place the object module in "a. out".

Compile the routines defined in "myprog1. C", "myprog2. C", and "myprog3. C". Once they are all compiled, link them into the object module "a. out".

```
% CC -c myprog1. C<CR>
% CC -c myprog2. C<CR>
% CC -c myprog3. C<CR>
% ld myprog1. o myprog2. o myprog3. o -lc<CR>
```

```
% dbx a. out<CR>
```

Run a program under the interactive debugger on the current site where the *a. out* file is from the C++ compiler.

```
% CC myprog. C -lsubs<CR>
```

Compile the routine defined in "myprog. C" and link it to the additional library "libsubs. a".

DOCUMENT CODE: UNIX-40604B

DATE REVISED: September 7, 1995

NOTES