Getting Started with
Gardens Point Component Pascal
Version 1.2.0 JVM (*September 2002*)

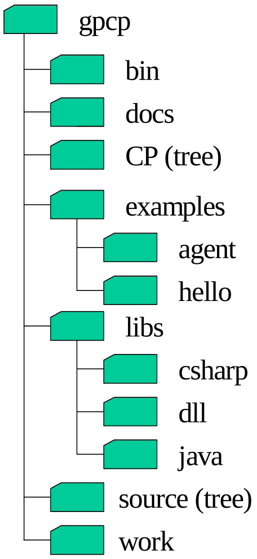
1. Introduction

Gardens Point Component Pascal (***gpcp***) is an implementation of the Component Pascal Language, as defined in the Component Pascal Report from Oberon Microsystems. It is intended that this be a faithful implementation of the report, except for those changes that are explicitly detailed in the release notes. Any other differences in detail should be reported as potential bugs.

The compiler produces either Microsoft.NET intermediate language or Java byte-codes as output. The compiler can be bootstrapped on either platform. These notes refer to the Sun Microsystems JVM platform.

Details on the specifics of this implementation of Component Pascal are found in the release notes that come with the distribution.

2. Installing and Testing the Compiler



The compiler is distributed as a single zip file, which has the structure shown on the left. It is also available in as **gpcp.tar.gz** for Unix systems. From version 1.1.4 the compiler is also available as a windows installer file.

The archive is typically expanded into a root directory named **`gpcp`** and has seven subdirectories. These include the binary files of the compiler, the documentation, the program examples, the library symbol files, and the source of the compiler.

This section describes the steps required to install and try out the compiler.

Environment
The compiler requires a Java Runtime Environment which implements the runtime for Java 2. All testing of the compiler has been done using Sun Microsystems JDK versions 1.2 or 1.3, or in some cases version 1.1.7.

The distribution

The seven subdirectories of the distribution are –

bin	the binary and shell/batch files of the compiler
CP	the class file tree of the compiler and tools
docs	the documentation, including this file
examples	some example programs
libs	contains the symbol files for the library modules
source	the compiler source files
work	a working directory to play around with

The **bin** directory needs to be on your PATH, and the environment variable CPSYM must point to the **libs** directory. Typical commands to set these variables are –

```
set CPSYM=.;C:\gpcp\libs
set PATH=%PATH%;C:\gpcp\bin
```

On Unix or similar systems the commands would be

```
CPSYM=.:$CPROOT/gpcp/libs
PATH=$PATH:$CPROOT/gpcp/bin
```

Where *CPROOT* is an environment variable that points to the root of the installation.

If you use the Windows installer version, simply run the `setup.exe` file, and respond to the prompts. The installer will modify the paths as necessary.

Running your first program

Go to the work directory. With your favorite editor create the file (say) `hello.cp.`

```
MODULE Hello;
  IMPORT CPmain, Console;
BEGIN
  Console.WriteString(“Hello CP World”);
  Console.WriteLn;
END Hello.
```

Make sure that the `CPSYM` environment variable includes the `gpcp/libs` directory, and that `gpcp/bin` is on the executable path.

From the command line, type

```
> cprun gpcp hello.cp
```

the system should respond …

```
Generated: CP/Hello/Hello.class
#gpcp: <Hello> no errors
> ▯
```

The files `Hello.cps` and `CP/Hello/Hello.class` should have been created in the working directory.

In order for this program to run, it must have access to the facilities of the CP runtime system. These facilities will be found in the class tree rooted at `gpcp/CP`, which is on the suggested classpath.

You may now run the program by the command “`cprun Hello`”.

The examples

The example programs are in sub-directories under the *examples* directory. The folder **hello** holds some simple command line programs. *HelloWorld.cp* is an elaborate version of the "hello world" canonical program. *Nqueens.cp* is a recursive backtracking version of the N-Queens problem solved for all board sizes from 8 to 13. *Hennessy.cp* is a version of the Hennessy integer benchmarks.

A file *README.txt* gives instructions for compiling and running the programs.

3. Browsing Modules

The *Browse* tool has been included with this release. This tool can show the exported interface for modules in either text or html format. Details on the use of this tool can be found in the Release Notes.

4. Reporting Bugs

If you find a bug

If you find what you believe is a bug, please send a report to gpcp@qut.edu.au with the detail of the event. It would be particularly helpful if you can send the code of the shortest program that can illustrate the error.

If the compiler crashes

The compiler has an outer-level exception rescue clause (you can see this in the body of procedure **`CPascal.Compile()`**) which catches any exceptions raised during any per-file compilation attempt. The rescue code displays a **"<<compiler panic>>"** message on the console, and attempts to create a listing in the usual way. In most cases the rescue clause will be able to build an error message from the exception call chain, and will send this both to the screen and to the listing file.

In almost all cases, the compiler panic will be caused by failed error recovery in the compiler, so that the other error messages in the listing will point to the means of programming around the compiler bug. Nevertheless, it is important to us to remove such bugs from the compiler, so we encourage users who turn up error of this kind to send us a listing of a (hopefully minimal) program displaying the phenomenon.

In order to see how such a rescue clause works, here is an example of a program which deliberately causes a runtime error. When the program is run, the error is caught at the outer level and an error message is generated. After generating the error message, there is still the option of aborting the program with the standard error diagnostics. This is done by re-raising the same exception, and this time allowing the exception to propagate outwards to the invoking command line processor.

```
MODULE Crash;
  IMPORT CPmain, Console, RTS;
  TYPE Ptr = POINTER TO ARRAY OF CHAR;
  VAR  p : Ptr;
  PROCEDURE Catch;
  BEGIN
    P[0] := “a”;
  RESCUE (exc)  (* exc is of type RTS.NativeException *)
    Console.WriteString(“Caught Exception: ”);
    Console.WriteString(RTS.getStr(exc));
    Console.WriteLn;
   (* THROW(exc) *)
  END Catch;
BEGIN
  Catch
END Crash.
```

When this program is compiled and run, the following is the result –

```
   ➢ cprun gpcp Crash.cp
   ➢ Generated: CP/Crash/Crash.class
   ➢ #gpcp: <Crash> No errors
   ➢ cprun Crash
   Caught Exception: java/lang/NullPointerException
   ➢ ▯
```

If the detailed stack trace is required, the exception is re-raised by calling the non-standard built-in procedure **`THROW(ex).`** The comment in the source code shows where to place the call.

Posting to the Mail Group

The Gardens Point Modula-2 mail group can be used to discuss issues concerning the evolution of ***gpcp***. In order to join this mail group, send email to majordomo@dstc.qut.edu.au with a blank subject line and the words subscribe gpm in the body. The team will post notices regarding updates to that mail group.