Practical No. 10 Using Autotools

Autotools is a suite of programs for creating build systems for software projects. It involves creating several configuration files like configure.ac, Makefile.am, and a bootstrap script.

Step 1: Structure the C Project

Suppose your project is structured like this:

CSS file:

```
my_project/
├── src/
|   ├── main.c
|   ├── utils.c
|   └── utils.h
├── include/
|   └── my_project.h
├── Makefile.am
├── configure.ac
└── bootstrap.sh
```

Step 2: Write configure.ac

This file is used by autoconf to create a configure script. Create a configure.ac file in the root of your project directory.

Bash File:

```
AC_INIT([my_project], [1.0], [you@example.com])

AM_INIT_AUTOMAKE

AC_PROG_CC

AC_CONFIG_FILES([Makefile src/Makefile])

AC_OUTPUT
```

Step 3: Write Makefile.am

Create Makefile.am in the root directory:

Make File:

```
SUBDIRS = src
```

And another Makefile.am in the src directory:

```
bin_PROGRAMS = my_project

my_project_SOURCES = main.c utils.c

my_project_CFLAGS = -I$(top_srcdir)/include
```

Step 4: Write bootstrap.sh

This script will generate the configure script and make your project ready to build.

Bash File

```
#!/bin/sh

autoreconf --install
```

Make sure to give the bootstrap.sh script executable permissions:

```
chmod +x bootstrap.sh
```

Step 5: Execute the Build Process

Run the following commands in sequence:

1. Run the bootstrap script to generate the configure script:

   Bash file:

   Copy code

   ```
   ./bootstrap.sh
   ```

2. Run the configure script to check dependencies and create Makefiles:

   Bash File:

   Copy code

   ./configure
3. Build the project using make:

   Bash file:

   Copy code

   make
4. Run the generated binary:

   Bash file:

   Copy code

   ./src/my_project

Practical No. 10 Using CMake:

CMake is simpler and widely used for cross-platform projects. Here's how you can configure a project using CMake:

Step 1: Structure the C Project

Use the same project structure as before:

Css File:

```
my_project/
├── src/
|   ├── main.c
|   ├── utils.c
|   └── utils.h
├── include/
|   └── my_project.h
└── CMakeLists.txt
```

Step 2: Write CMakeLists.txt

In the root directory of your project, create a CMakeLists.txt file:

Cmake File:

```
cmake_minimum_required(VERSION 3.10)
project(my_project VERSION 1.0)


# Specify C standard
set(CMAKE_C_STANDARD 11)


# Include header files
include_directories(include)


# Add the executable
add_executable(my_project src/main.c src/utils.c)
```

Step 3: Execute the Build Process

1. Create a build directory and navigate into it:

   Bash File:
   ```
   mkdir build
   cd build
   ```

2. Run cmake to generate the build files:

   Bash File:
   ```
   Copy code
   cmake ..
   ```

3. Build the project using make:

   Bash File:
   ```
   Copy code
   make
   ```

4. Run the generated binary:

   Bash File:
   ```
   Copy code
   ./my_project
   ```

Explanation:

- cmake_minimum_required ensures a compatible CMake version.
- project defines the project name and version.
- include_directories specifies directories containing header files.
- add_executable specifies the output executable and source files.

Conclusion:

- **Autotools**: Good for projects that need to support a wide range of UNIX-like systems, but it involves more files and scripts.
- **CMake**: Easier to set up, cross-platform, and becoming more popular for C/C++ projects.