1. 引言  
    本程序构建了一个基础的命令行 Shell 程序，具备处理常见命令操作的能力，如基本的命令输入、目录切换、管道操作以及输入输出重定向等功能。
2. **函数剖析**  
   split函数  
   std::vector<std::string> split(const std::string &s, char delimiter)  
   用途：该函数旨在将给定的字符串 s 依据特定的分隔字符 delimiter 进行切分。其成果是将切分后的子字符串汇集于一个std::vector<std::string>中，并最终返回此向量。

main  函数  
int main()

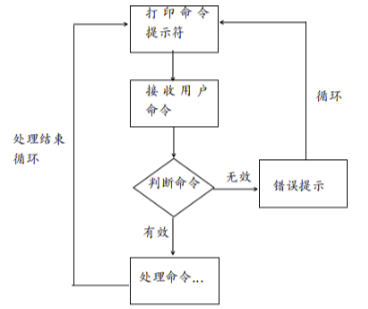
核心作用：作为程序的主入口点，主函数承载了 Shell 程序的关键逻辑流程。

1. **关键逻辑阐释**   
   **命令输入与处置机制**  
   借助一个永无止境的循环架构，持续向用户展示输入提示cppshell> ，并运用 std::getline 函数精准获取用户键入的字符串，将其存置于 input 变量之中。  
   倘若用户输入的是 exit 字符串，程序随即终止循环，实现退出操作。  
      
   **目录切换（ cd  命令）功能实现**  
   细致检测输入内容是否起始于 cd 。一旦符合此条件，便提取出紧随其后的路径信息，并调用 chdir 函数尝试切换当前工作目录。依据切换的成败结果，相应地输出明确的提示消息。

**命令切分与精细处理**  
 严谨审查输入中是否存有管道符号  |  。若存在，则将输入拆解为多个独立的命令，并妥善存储于 commands 向量当中。  
 针对每个单独的命令，进一步依照空格进行细分，将所得的参数逐一收纳至  args\_list  向量的对应位置。

**进程创建与高效执行**  
 借助 fork 函数创建子进程，以分别执行每一个命令。  
 在子进程内部，依据具体需求精心实施输入输出重定向操作。同时，运用 system 函数来确切执行命令。  
 处于父进程一侧，耐心等待子进程的完结，并精心打理管道的读写相关操作。

**整体业务流程图**



1. **重要提示与注意事项**  
    代码内部运用了一系列系统调用与库函数，例如  chdir  、 pipe  、 fork  、 execvp  等。为确保程序的正常运行，需在适配的操作系统环境中启动，并严格保障具备相应的操作权限。  
    在处理输入输出重定向以及管道操作时，务必谨慎对待文件描述符的各类操作，谨防资源泄露以及错误的滋生。