==> zadanie_1_01.c <==
Zobrazte do standardneho vystupu pocet vsetkych regularnych suborov v adresarovej strukture. Adresarova struktura moze byt specifikovana pomocou pociatocneho adresara. Tento adresar je specifikovany parametrom programu. V pripade ze je definovany prepinac "-s" alebo "--size", zobrazi sa velkost vsetkych regularnych suborov v adresarovej strukture v bytoch. Ak nie je specifikovany parameter pociatocneho adresara, vytvorte detsky proces, ktory od pouzivatela vypyta nazov pociatocneho adresara. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. Po spracovani pouzivatelom zadanej adresarovej struktury ma mat pouzivatel moznost pokracovat v spracovani inej adresarovej struktury.
Volanie programu: zadanie.elf [-s | --size] [<pociatocny_priecinok>]

==> zadanie_1_02.c <==
Zobrazte do standardneho vystupu mena vsetkych regularne subory v adresarovej strukture, ku ktorym bol pristup za menej ako <pocet_dni> dni. Adresarova struktura je specifikovana pomocou pociatocneho adresara. Tento adresar je specifikovany parametrom programu. Pocet dni je specifikovany parametrom programu. V pripade, ze je definovany prepinac "-c" alebo "--count", zobrazi sa len ich pocet. Ak nie je specifikovany parameter pociatocneho adresara, vytvorte detsky proces, ktory od pouzivatela vypyta nazov pociatocneho adresara. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf <pocet_dni> [-c | --count] [<pociatocny_priecinok>]

==> zadanie_1_03.c <==
Napiste program, ktory spusti program <prog> zadany parametrom "-p" alebo "--program" (bez argumentov). Program <prog> aj povodny program nech vypisu obsah suboru <file> (zadaneho volitelnym parametrom "-f" alebo "--file") na standardny vystup. Pritom oba vypisy musia byt koordinovane signalmi, aby sa vzajomne nemiesali: riadky vypisu sa budu pravidelne striedat podla uvedeneho vzoru vystupu. Ak nie je zadany argument <file>, vytvorte detsky proces, ktory na standardny vystup vypise zoznam vsetkych regularnych suborov v aktualnom priecinku (rekurzivne do hlbky); a tento proces zabezpeci nacitanie volby pouzivatela vzhladom na parameter "-f". Rodicovsky proces nech zatial caka na spracovanie vstupu od pouzivatela detskym procesom. Identifikatorom procesov na vystupe nech je ich PID. Program musi prehladavat aktualny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf -p <prog> [-f <file>]
Priklad vystupu: <proces1>: <1. riadok suboru <file>>
<proces2>: <1. riadok suboru <file>>
<proces1>: <2. riadok suboru <file>>
<proces2>: <2. riadok suboru <file>>
...

==> zadanie_1_04.c <==
Napiste program, ktory v zadanom adresari, uvedenom ako argument, a jeho podadresaroch, najde vsetky symbolicke linky a vypise ich zoznam na standardny vystup. Ak bude program spusteny s prepinacom "-e"/"--exist", vyhlada len pocet liniek s existujucim cielovym suborom; s prepinacom "-n" alebo "--not_exist" len pocet liniek s neexistujucim cielovym suborom. Ak nie je specifikovany parameter pociatocneho adresara, vytvorte detsky proces, ktory od pouzivatela vypyta nazov pociatocneho adresara. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf [-e|--exist | -n | --not_exist] [<pociatocny_priecinok>]

==> zadanie_1_05.c <==
Napiste program, ktory zobrazi na obrazovku riadky suboru <file>, ktore obsahuju vzor <mask> dany argumentom "-m" alebo "--mask" (emulacia grepu) programu. Meno suboru moze byt dane argumentom programu. V pripade, ze je zadany prepinac "-c" alebo "--count", vypise sa na obrazovku pocet riadkov, ktore splnaju podmienku vzoru. Program nesmie pouzivat na hladanie vzoru externy program. Ak nie je zadany argument <file>, vytvorte detsky proces, ktory na standardny vystup vypise zoznam vsetkych regularnych suborov v aktualnom priecinku (rekurzivne do hlbky); a tento proces zabezpeci nacitanie volby pouzivatela vzhladom na vstupny subor <file>. Rodicovsky proces nech zatial caka na spracovanie vstupu od pouzivatela detskym procesom. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf -m <mask> [-c] [<file>]

==> zadanie_1_06.c <==s
Zobrazte do standardneho vystupu zoznam vsetkych poloziek adresara v adresarovej strukture (emulacia "ls -ogR"). Vstupny bod adresarovej struktury je definovany pociatocnym adresarom <pociatocny_priecinok>. Ten moze byt dany ako argument programu. Do vystupu zabrazte len mena suborov (bez cesty k suboru). V pripade ze je zadany prepinac "-i" alebo "--id", zobrazi sa aj uid vlastnika a gid skupiny suborov spolu so zoznamom (emulacia "ls -nR"). Ak nie je specifikovany parameter pociatocneho adresara, vytvorte detsky proces, ktory od pouzivatela vypyta nazov pociatocneho adresara. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf [-i | --id] [<pociatocny_priecinok>]

==> zadanie_1_07.c <==
Napiste program, ktory spusti program <prog> zadany (bez argumentov) parametrom "-p" alebo "--prog". Program <prog> musi vypisovat na standardny vystup obsah aktualneho adresara (tak ako prikaz "ls ."). Ak je specifikovany pomocou prepinaca "-l" alebo "--logfile" subor <logfile>, vystup programu sa vypisuje do neho, inak na standardny vystup. Ak nie je zadany argument <prog>, vytvorte detsky proces, ktory na standardny vystup vypise zoznam vsetkych spustitelnych suborov v aktualnom priecinku (rekurzivne do hlbky); a tento proces zabezpeci nacitanie volby pouzivatela vzhladom na parameter <prog>. Rodicovsky proces nech zatial caka na spracovanie vstupu od pouzivatela detskym procesom. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. V pripade, ze je zadany prepinac "-c" alebo "--copy", standardny vstup z klavesnice terminalu sa skopiruje na standardny vystup a zaroven do suboru <logfile>.
Volanie programu: zadanie.elf [-c | --copy] [-p <prog>] [-l <logfile>]

==> zadanie_1_08.c <==
Napiste program, ktory bude manipulovat so subormi. Po jeho spusteni sa zobrazi menu:

1) Vytvorit adresar
2) Kopirovat subory do adresara
3) Zalohovat subory z aktualneho adresara
4) Zobrazit obsah adresara
5) Koniec

Do adresara, ktory bude vytvoreny, sa v pripade volieb 2 a 3 budu subory ukladat tak, aby sa v pripade, ak uz jestvuju, neprepisovali, ale aby sa automaticky archivovali rozsirenim mena suboru o koncovku <cislo_verzie>. V pripade volby 2 je nutne od pouzivatela ziskat zoznam suborov, ktore sa maju kopirovat (relativne alebo absolutne cesty). Pri volbe 4 je nutne vypisovat obsah adresara rekurzivne do hlbky pomocou API funkcii jadra systemu. Rozhranie s pouzivatelom musi zabezpecovat rodicovsky proces, volby 1 az 4 detske procesy asynchronne voci rodicovskemu. Program musi vediet spracovat volitelny parameter "-c" alebo "--count" s povinnym ciselnym argumentom. Ak je zadany tento parameter, pouzivatel moze maximalne --count krat spustit funkcie 1 az 4.
Volanie programu: zadanie.elf

==> zadanie_1_09.c <==
Zobrazte do standardneho vystupu zoznam vsetkych adresarov v adresarovej strukture, ktore boli vytvorene pred viac ako <pocet_dni> dnami. Adresarova struktura je specifikovana pomocou pociatocneho adresara <pociatocny_priecinok>, ktory je zadany ako posledny argument programu. Pocet dni <pocet_dni> je tiez argumentom programu, ale specifikovany pomocou volitelneho prepinaca "-d" alebo "--days". V pripade, ze je definovany prepinac "-c" alebo "--count", zobrazi sa celkovy pocet adresarov (teda aj tych, ktore podmienke nevyhovuju) a pocet adresarov splnujucich danu podmienku. Ak nie je specifikovany parameter <pocet_dni>, vytvorte detsky proces, ktory ho od pouzivatela nacita. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. Volanie programu: zadanie.elf [-d <pocet_dni>] [-c] <pociatocny_priecinok>

==> zadanie_1_10.c <==
Zobrazte do standardneho vystupu zoznam vsetkych regularnych suborov v adresarovej strukture. Adresarova struktura je specifikovana pomocou pociatocneho adresara <pociatocny_priecinok>. Tento adresar je specifikovany volitelnym argumentom programu. V pripade, ze je definovany prepinac "-c" alebo "--count", zobrazi sa pocet regularnych suborov namiesto zoznamu suborov. Ak nie je specifikovany parameter <pociatocny_priecinok>, vytvorte detsky proces, ktory ho od pouzivatela nacita. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. Po spracovani pouzivatelom zadanej adresarovej struktury ma mat pouzivatel moznost pokracovat v spracovani inej adresarovej struktury.
Volanie programu: zadanie.elf [-c | --count] [<pociatocny_priecinok>]

==> zadanie_1_11.c <==
Zobrazte do standardneho vystupu zoznam vsetkych podadresarov v adresarovej strukture abecedne usporiadanych (tip: skuste na usporiadanie vyuzit napriklad funkciu "sort" zo standardnej kniznice jazyka C). Adresarova struktura je specifikovana pomocou pociatocneho adresara <pociatocny_priecinok>. Tento adresar je specifikovany povinnym prvym parametrom programu. V pripade, ze je definovany prepinac "-r" alebo "--reverse", zobrazia sa adresare v opacnom poradi. Program nesmie pouzit volanie externeho programu. Po spracovani pouzivatelom zadanej adresarovej struktury ma mat pouzivatel moznost pokracovat v spracovani inej adresarovej struktury. Ak sa pouzivatel rozhodne zobrazit podadresare z dalsej adresarovej struktury, vytvorte detsky proces, ktory od pouzivatela nacita potrebne udaje. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf <pociatocny_priecinok> [-r | --reverse]

==> zadanie_1_12.c <==
Napiste program, ktory vytvori dva detske procesy. Oba detske procesy nech vypisu obsah suboru <file> (zadaneho volitelnym parametrom "-f" alebo "--file") na standardny vystup. Pritom oba vypisy musia byt koordinovane signalmi, aby sa vzajomne nemiesali: riadky vypisu sa budu pravidelne striedat podla uvedeneho vzoru vystupu. Ak nie je zadany argument <file>, vytvorte dalsi detsky proces, ktory na standardny vystup vypise zoznam vsetkych regularnych suborov v aktualnom priecinku (rekurzivne do hlbky); a tento proces zabezpeci nacitanie volby pouzivatela vzhladom na parameter "-f". Rodicovsky proces nech zatial caka na spracovanie vstupu od pouzivatela detskym procesom. Identifikatorom procesov na vystupe nech je ich PID. Program musi prehladavat aktualny adresar rekurzivne pomocou API funkcii jadra systemu. Volanie programu: zadanie.elf [-f <file>]
Priklad vystupu:
<proces1>: <1. riadok suboru <file>>
<proces2>: <1. riadok suboru <file>>
<proces1>: <2. riadok suboru <file>>
<proces2>: <2. riadok suboru <file>>
...

==> zadanie_1_13.c <==
Zobrazte do standardneho vystupu zoznam vsetkych prazdnych priecinkov v adresarovej strukture. Zoznam ma obsahovat na kazdom riadku absolutnu cestu priecinka, ktory vyhovuje podmienke zadania. Adresarova struktura je specifikovana pomocou pociatocneho adresara <pociatocny_priecinok>. Tento adresar je specifikovany povinnym argumentom programu. V pripade, ze je definovany prepinac "-m" alebo "--match", zobrazi sa pocet prazdnych priecinkov. Ak je zadany prepinac "-n" alebo "--no_match", zobrazi sa pocet priecinkov, ktore nie su prazdne. Ak je zadany prepinac "-a" alebo "--all", zobrazi sa pocet vsetkych spracovanych priecinkov. Ak je zadana aspon jedna z troch volieb "-m", "-n", "-a", nezobrazuje sa zoznam prazdnych priecinkov. Po spracovani pouzivatelom zadanej adresarovej struktury ma mat pouzivatel moznost pokracovat v spracovani inej adresarovej struktury. Ak sa pouzivatel rozhodne pokracovat v cinnosti programu, vytvorte detsky proces, ktory od pouzivatela nacita potrebne udaje pre dalsiu cinnost. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf [-m | --match] [-n | --no_match] [-a | --all] <pociatocny_priecinok>

==> zadanie_1_14.c <==
Napiste program, ktory v adresarovej strukture, urcenej prvym argumentom programu <pociatocy_priecinok>, najde vsetky symbolicke linky a vypise ich zoznam na standardny vystup. Ak bude program spusteny s prepinacom "-e"/"--exist", vyhlada len pocet liniek s existujucim cielovym suborom; s prepinacom "-n" alebo "--not_exist" len pocet liniek s neexistujucim cielovym suborom. Po spracovani pouzivatelom zadanej adresarovej struktury ma mat pouzivatel moznost pokracovat v spracovani inej adresarovej struktury. Ak sa pouzivatel rozhodne pokracovat v cinnosti progeamu, vytvorte detsky proces, ktory od pouzivatela vypyta nazov dalsieho pociatocneho adresara. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi pouzivat volania opendir, readdir a closedir.
Volanie programu: zadanie.elf <pociatocny_priecinok>  [-e|--exist | -n | --not_exist]

==> zadanie_1_15.c <==
Napiste program, ktory na zaciatku ponukne pouzivatelovi zoznam vsetkych regularnych nespustitelnych suborov v aktualnom priecinku (vratane podpriecinkov - program musi prehladavat aktualny adresar rekurzivne pomocou API funkcii jadra systemu). Ked si jeden z nich pouzivatel zvoli, vytvoria sa dva detske procesy. Procesy budu zapisovat obsah pouzivatelom zvoleneho suboru do suboru, ktory bude specifikovany povinnym argumentom programu <output>. Na zadanie argumentu <output> pouzite povinny prepinac "-o" alebo "--output". Do suboru moze zapisovat vzdy len jeden proces. V jednom cykle zapisu sa zo zdrojoveho suboru precita prave jeden riadok textu. Zapis do suboru koordinujte rodicovskym procesom zasielanim signalov. Identifikatorom procesov vo vystupnom subore nech je ich PID.
Volanie programu: zadanie.elf –o <output>
Priklad obsahu vystupneho suboru:<proces1>: <1. riadok vstupneho suboru>
<proces2>: <2. riadok vstupneho suboru>
<proces1>: <3. riadok vstupneho suboru>
<proces2>: <4. riadok vstupneho suboru>

==> zadanie_1_16.c <==
Napiste program, ktory v zadanej adresarovej strukture najde vsetky regularne nespustitelne subory a prvy riadok kazdeho z nich zapise do vystupneho suboru <output> specifikovaneho volitelnym parametrom programu "-o" alebo "--output". Adresarova struktura je specifikovana pomocou volitelneho pociatocneho adresara <pociatocny_priecinok>, ktory je zadany argumentom programu. Ak nie je specifikovany parameter <pociatocny_priecinok>, vytvorte detsky proces, ktory ho od pouzivatela nacita. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. Ak nie je zadany arg. <output>, 1. riadok kazdeho spracovaneho suboru sa vypise na stdout.
Volanie programu: zadanie.elf [-o <output>] [<pociatocny_priecinok>]

==> zadanie_1_17.c <== (vysvetlit obmedzenie retval child procesu; vyriesit tento problem)
Napiste program, v ktorom sa  vytvori detsky proces. Detsky proces zobrazi do standardneho vystupu zoznam vsetkych podadresarov v priecinku, ktory je specifikovany argumentom programu <pociatocny_priecinok>. Hlbka rekurzivneho vnorenia je dana volitelnym argumentom "-l"/"--level". Ak nie je specifikovana, spracuva sa cely strom adresarov bez obmedzenia. Pocet adresarov nech zobrazi rodicovsky proces na konci vypisu. Vypisy koordinujte signalom SIGCHLD (vznikne po skoceni detskeho procesu) a poskytnutie informacie o pocte priecinkov navratovou hodnotou detskeho procesu. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. Po spracovani pouzivatelom zadanej adresarovej struktury ma mat pouzivatel moznost pokracovat v spracovani inej adresarovej struktury.
Volanie programu: zadanie.elf [-l <hlbka>] <pociatocny_priecinok>

==> zadanie_1_18.c <==
Napiste program, v ktorom sa  vytvori detsky proces. Rodicovsky proces zobrazi do standardneho vystupu zoznam vsetkych regularnych suborov v priecinku, ktory je specifikovany argumentom programu <pociatocny_priecinok>. Hlbka rekurzivneho vnorenia je dana volitelnym argumentom "-l"/"--level". Ak nie je specifikovana, spracuva sa iba pociatocny priecinok bez rekurzie. Pocet suborov nech zobrazi detsky proces na konci vypisu. Vypisy koordinujte pouzivatelskym signalom. V pripade rekurzivneho prehladavania musite pouzit API funkcie jadra pre pracu so suborovym systemom. Po spracovani zadanej adresarovej struktury ma mat pouzivatel moznost pokracovat v spracovani inej adresarovej struktury.
Volanie programu: zadanie.elf [-l <hlbka>] <pociatocny_priecinok>

==> zadanie_1_19.c <==
Zobrazte do standardneho vystupu zoznam vsetkych regularnych suborov v adresarovej strukture. Zoznam obsahuje na kazdom riadku plnu (absolutnu) cestu k suboru. Ak je specifikovany prepinac "-a" alebo "--after", zobrazia sa iba subory, ktore boli spristupnene po case <time>. Ak je specifikovany prepinac "-b" alebo "--before", zobrazia sa iba subory, ktore boli spristupnene pred casom <time>. Adresarova struktura je specifikovana pomocou pociatocneho adresara. Tento adresar je specifikovany volitelnym argumentom programu <pociatocny_priecinok>. Ak nie je specifikovany argument pociatocneho adresara, vytvorte detsky proces, ktory od pouzivatela vypyta nazov pociatocneho adresara. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf [-a <time> | -b <time>] [<pociatocny_priecinok>]

==> zadanie_1_20.c <==
Napiste program, ktory spusti programy <prog1> a <prog2> zadane parametrami "--program1" a "--program2" (bez argumentov). Programy <prog1> aj <prog2> nech vypisuju obsah suboru <file> (zadaneho volitelnym parametrom programu "-f" alebo "--file") na standardny vystup. Pritom oba vypisy musia byt koordinovane signalmi, aby sa vzajomne nemiesali: riadky vypisu sa budu pravidelne striedat podla uvedeneho vzoru vystupu. Ak nie je zadany argument <file>, vytvorte detsky proces, ktory na standardny vystup vypise zoznam vsetkych regularnych suborov v aktualnom priecinku (rekurzivne do hlbky); a tento proces zabezpeci nacitanie volby pouzivatela vzhladom na parameter "-f". Rodicovsky proces nech zatial caka na spracovanie vstupu od pouzivatela detskym procesom. Identifikatorom procesov na vystupe nech je ich PID. Program musi prehladavat aktualny adresar rekurzivne pomocou API funkcii jadra systemu.
Volanie programu: zadanie.elf --program1 <prog1> --program2 <prog2> [-f <file>]
Priklad vystupu: <proces1>: <1. riadok suboru <file>>
<proces2>: <1. riadok suboru <file>>
<proces1>: <2. riadok suboru <file>>
<proces2>: <2. riadok suboru <file>>
...
==> zadanie_1_21.c <=
Napiste program, ktory v zadanej adresarovej strukture najde vsetky spustitelne subory prikazoveho interpreta bash a ich zoznam zapise do vystupneho suboru <output> specifikovaneho volitelnym prepinacom "-o" alebo "--output". Zoznam obsahuje absolutne cesty k jednotlivym skriptom. Adresarova struktura je specifikovana pomocou volitelneho pociatocneho adresara <pociatocny_priecinok>, ktory je zadany argumentom programu. Ak nie je specifikovany parameter <pociatocny_priecinok>, vytvorte detsky proces, ktory ho od pouzivatela nacita. Rodicovsky proces nech caka na spracovanie vstupu od pouzivatela detskym procesom. Po spracovani vstupu od pouzivatela da detsky proces rodicovskemu pomocou signalu vediet, ze moze pokracovat vo vykonavani cinnosti. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. Ak nie je zadany argument <output>, zoznam sa vypise na standardny vystup.
Volanie programu: zadanie.elf [-o <output>] [<pociatocny_priecinok>]

==> zadanie_1_22.c <==
Zobrazte do standardneho vystupu zoznam vsetkych podadresarov v adresarovej strukture usporiadanych podla poctu suborov, ktore obsahuju (tip: skuste na usporiadanie vyuzit napriklad funkciu "sort" zo standardnej kniznice jazyka C). Do poctu sa pocitaju iba subory priamo v danom priecinku, nie v jeho podpriecinkoch. Adresarova struktura je specifikovana pomocou pociatocneho adresara <pociatocny_priecinok>. Tento adresar je specifikovany povinnym prvym parametrom programu. V pripade, ze je definovany prepinac "-r" alebo "--reverse", zobrazia sa adresare v opacnom poradi. Program nesmie pouzit volanie externeho programu. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. Spracovanie adresarovej struktury musi byt vykonane v procese rodica, spocitanie poctu suborov v procese potomka. Zvazte moznost urychlenia spracovania adresarovej struktury pomocou asynchronnej cinnosti potomkov vzhladom na rodica.
Volanie programu: zadanie.elf <pociatocny_priecinok> [-r | --reverse]

==> zadanie_1_23.c <==
Napiste program, ktory v zadanej adresarovej strukture najde vsetky regularne nespustitelne subory a poslednych <pocet> riadkov kazdeho z nich zapise na standardny vystup. Ak nie je pouzity prepinac "-c", zapise sa iba posledny riadok. Adresarova struktura je specifikovana pomocou pociatocneho adresara <pociatocny_priecinok>. Prehladavanie adresarovej struktury robte pomocou API funkcii jadra systemu. Vystup zabezpecia dva procesy (rodic a potomok) podla ukazky. Rodic zapisuje uplnu cestu spracovavaneho suboru, potomok pozadovane riadky.
Volanie programu: zadanie.elf [-c <pocet>] <pociatocny_priecinok>
Ukazka vystupu pre argument "-c 2": <cesta_suboru_1>
<predposledny riadok>
<posledny riadok>
<cesta_suboru_2>
<predposledny_riadok>
<posledny_riadok>
...

==> zadanie_1_24.c <==
Napiste program, ktory v zadanej adresarovej strukture najde vsetky spustitelne subory a na standardny vystup vypise zoznam tych suborov, ktore obsahuju retazec podla zadanej masky <mask> prepinaca "-m" (emulacie prikazu grep). Zoznam na kazdom riadku obsahuje absolutnu cestu suboru, ktory vyhovuje podmienke. Adresarova struktura je specifikovana pomocou pociatocneho adresara <pociatocny_priecinok>. Prehladavanie adresarovej struktury robte v procese rodica pomocou API funkcii jadra systemu. Overenie, ci dany spustitelny subor obsahuje retazec podla zadanej masky, robte v procese potomka. Zvazte moznost urychlenia spracovania adresarovej struktury pomocou asynchronnej cinnosti potomkov vzhladom na rodica (tip: je ale vhodne, ak je spracovanie vystupu synchronizovane; preco?)
Volanie programu: zadanie.elf -m <mask> <pociatocny_priecinok>

==> zadanie_1_25.c <==
Zobrazte do standardneho vystupu zoznam vsetkych podadresarov v adresarovej strukture usporiadanych podla poctu suborov, ktore obsahuju (tip: skuste na usporiadanie vyuzit napriklad funkciu "sort" zo standardnej kniznice jazyka C). Do poctu sa pocitaju aj subory z jeho podpriecinkov. Adresarova struktura je specifikovana pomocou pociatocneho adresara <pociatocny_priecinok>. Tento adresar je specifikovany povinnym prvym parametrom programu. V pripade, ze je definovany prepinac "-r" alebo "--reverse", zobrazia sa adresare v opacnom poradi. Program nesmie pouzit volanie externeho programu. Program musi prehladavat pociatocny adresar rekurzivne pomocou API funkcii jadra systemu. Spracovanie adresarovej struktury musi byt vykonane v procese rodica, spocitanie poctu suborov v procese potomka. Zvazte moznost urychlenia spracovania adresarovej struktury pomocou asynchronnej cinnosti potomkov vzhladom na rodica.
Volanie programu: zadanie.elf <pociatocny_priecinok> [-r | --reverse]