Oznaka poročila: ARRS-RPROJ-ZP-2011-1/162

ZAKLJUČNO POROČILO O REZULTATIH RAZISKOVALNEGA PROJEKTA

A. PODATKI O RAZISKOVALNEM PROJEKTU

1. Osnovni podatki o raziskovalnem projektu

Šifra projekta	J2-9311	
Naslov projekta	Modeliranje transporta tekočine v nano cevkah	
Vodja projekta	5570	Ivo Kljenak
Tip projekta	J	Temeljni projekt
Obseg raziskovalnih ur	3.150	
Cenovni razred	C	
Trajanje projekta	07.2007 - 06.2010	
Nosilna raziskovalna organizacija	106	Institut "Jožef Stefan"
Raziskovalne organizacije - soizvajalke		
Družbeno-ekonomski cilj	13.	Splošni napredek znanja - RiR financiran iz drugih virov (ne iz splošnih univerzitetnih fondov - SUF)

1.1. Družbeno-ekonomski cilj[1]

Šifra	13.02
Naziv	Tehnološke vede - RiR financiran iz drugih virov (ne iz SUF)

2. Sofinancerji[2]

1.	Naziv	
	Naslov	
2.	Naziv	
	Naslov	
3.	Naziv	
	Naslov	

B. REZULTATI IN DOSEŽKI RAZISKOVALNEGA PROJEKTA

3. Poročilo o realizaciji programa raziskovalnega projekta[3]

Projekt se je pričel z razvojem modela za Monte-Carlo velekanonično simulacijo (Grand Canonical Monte-Carlo simulation) adsorpcije dvokomponentne plinske mešanice v enostenski ogljikovi nanocevki. Večina dosedanjih del v literaturi iz tega področja obravnava mešanico CH_4 (metana) in nekega drugega plina. Razviti model je bil uporabljen za simulacijo adsorpcije ekvimolarnih mešanic H_2/N_2 (vodik-dušik) in H_2/O_2 (vodik-kisik). Simulacij adsorpcije teh mešanic doslej še nismo zasledili v literaturi. Adsorbirane molekule so bile modelirane kot sferični delci. Rezultati simulacij v literaturi so namreč pokazali, da tovrstna poenostavitev ne vpliva bistveno na simulacijo obnašanja plinov, ki smo jih obravnavali (v poznejši fazi tudi CO_2). Pri sami simulaciji to pomeni, da se sile med posameznimi atomi znotraj molekul in rotacije molekul ne obravnavajo. Interakcija med delci je bila modelirana s pomočjo Lennard-Jonesovega potenciala. Interakcijski potenciali med različnimi komponentami so bili definirani z uporabo kombinacijskih pravil Lorentz-Berthelot. Model je bil preverjen s primerjavo rezultatov, dobljenih za eno samo komponento, s teoretičnimi rezultati iz literature.

Specifičnost obravnavanih mešanic je v tem, da imata v Lennard-Jonesovih potencialih za komponente N_2 in H_2 skoraj enaki globini ("well depths"), medtem ko imata O_2 in H_2 skoraj enaka premera lokacij trkov ("site collision diameters"). Adsorpcija je bila omejena na notranjost nanocevk. Obravnavane so bile nanocevke petih različnih premerov:
— nanocevke (8,8) premera 1,085 nm,
— nanocevke (16,16) premera 2,170 nm,
— nanocevke (24,24) premera 3,255 nm,
— nanocevke (32,32) premera 4,340 nm,
— nanocevke (40,40) premera 5,425 nm.

Nanocevke so bile obravnavane kot toge. Potenciali adsorbent-nanocevka so bili sumirani po parih, ki jih tvorijo molekule adsorbenta in atomi ogljika iz nanocevk. Pri simulacijah je bil obravnavan kontrolni volumen, ki vsebuje eno samo nanocevko, s periodičnimi robnimi pogoji v osni smeri. Predpostavljena je bila t.i. sobna temperature (25 ^oC), medtem ko je tlak zavzemal vrednosti od 10 do 1000 bar.

Rezultati simulacij so pokazali sledeče:
— adsorpcijska selektivnost se veča s tlakom,
— nanocevke so selektivne za H_2 v mešanici H_2/N_2, in za O_2 v mešanici H_2/O_2,
— pri mešanici H_2/O_2 je selektivnost opazna pri nižjih tlakih kot pri mešanici H_2/N_2.

Opravljene simulacije so potrdile domnevo o selektivnosti nanocevk pri adsorpciji in so predstavljale zasnovo za nadaljnji razvoj modela transporta tekočine (plina ali kapljevine) v nanocevkah.

Razviti model je bil nato dopolnjen z določevanjem kemičnega potenciala komponent mešanice s kanonično simulacijo in uporabo Widomove metode testnega delca.

Model je bil nato razširjen na simulacijo adsorpcije trokomponentne plinske mešanice, z dodatnim upoštevanjem gibanja posameznih molekul zaradi mesebojnega vpliva in vpliva nanocevk, pri ravnovesnih pogojih. Gibanje posameznih molekul je določeno z integracijo gibalnih enačb, pri čemer so sile dobljene iz istih potencialov, kot v velekanonični simulaciji. Trki molekul ob notranjost nanocevk so modelirani kot popoloma elastični. Pred pričetkom simulacije toka je potrebno določiti ravnovesno konfiguracijo z velekanonično Monte-Carlo simulacijo. Proces uravnovešenja implicitno simulira adsorpcijo plina na steni nanocevke. V literaturi so do sedaj simulacije toka plinov skozi nanocevke omejene na dvokomponentne mešanice, medtem ko so simulacije obnašanja trokomponentnih mešanic v majhnih prostorih (na nano skali) omejene na obnašanje v nanoporah zeolitov. Tako kot pri dvokomponentni mešanici, je bila interakcija med molekulami modelirana s pomočjo Lennard-Jonesovega potenciala, interakcijski potenciali med različnimi komponentami definirani z uporabo kombinacijskih pravil Lorentz-Berthelot, in potenciali med nanocevkami in adsorbiranim plinom določeni s seštevanjem prispevkov posameznih parov, ki jih tvorijo molekule plinov in atomi nanocevk.

Model je bil apliciran na simulacijo mešanice plinov $O_2/N_2/CO_2$, v kateri sta O_2 in N_2 v enakem razmerju, kot v zraku. Obravnavane so bile naslednje nanocevke: (10,10), (16,16) in (24,24), s premeri 1,356 nm, 2,170 nm in 3,255 nm. Simulacije so bile izvedene z računskim volumnom, ki vsebuje eno samo nanocevko s periodičnimi robnimi pogoji v osni smeri. Predpostavljena je bila t.i. sobna temperatura (298 K), medtem ko je tlak zavzemal vrednosti od 1 do 100 bar. Izbira pogojev tlaka in temperature je bila motivirana s potencialno aplikacijo simuliranega pojava v procesni tehniki.

Rezultati simulacij so pokazali, da se v najtanjši nanocevki, premera 1,356 nm, molekule plina zbirajo na razdalji približno 0,35 nm od stene in v območju osi cevke. V cevki premera 2,170 nm se molekule zbirajo v plasti, oddaljeni približno 0,35 nm od stene cevi, ter so nato enakomerno razporejene znotraj namišljenega koncentričnega kroga, ki je oddaljen približno 0,7 nm od stene cevke. Podoben vzorec je nastal pri simulaciji obnašanja plina v najdebelejši nanocevki, premera 3,255 nm. S spreminjanjem tlaka se spreminja tudi gostota molekul, medtem ko je osnovni vzorec razporejanja nespremenjen. Analize so pokazale, da se z večanjem tlaka v cevkah adsorbira čedalje manjši delež CO_2 (ki je tudi vedno manjši, kot je delež CO_2 v osnovni mešanici). Rezultati so torej:
— potrdili adsorpcijsko selektivnost nanocevk,
— pokazali potencialno možnost uporabe nanocevk za ločevanje CO_2 iz zraka.

Model je bil končno razširjen za simulacijo toka plina z uporabo neravnovesne molekularne dinamike. Po ustvarjanju začetne konfiguracije z velekanonično simulacijo se tok mešanice sproži z ustvarjanjem višje molekularne gostote plina v enem koncu cevi. Na ta način je simuliran tok plina zaradi tlačne razlike. Izdelani model omogoča opazovanje ločevanja posameznih komponent plinske mešanice pri pretoku skozi nanocevke, ter lahko predstavlja osnovo za preučevanje možnosti uporabe nanocevk za razvoj separacijskih membran v procesni tehniki.

Opisani model sestoji iz dveh lastnih računalniških programov:
1. Program NANOFLOW.C za simulacijo transporta plina. Program, ki je bil razvit samostojno, omogoča simulacijo dveh procesov:
— adsorpcije plina na steni nanocevke za določitev začetne konfiguracije,
— toka plina skozi nanocevko.
2. Program NANOTUBE.C za določitev koordinat ogljikovih atomov, ki tvorijo nanocevko z določenimi karakteristikami. Program je bil razvit na osnovi podobnega programa iz literature.

Izpisa obeh računalniških programov sta podana kot priponki temu poročilu.

Specifičnost projekta je bila v tem, da se uporabljajo izsledki fizikalnih raziskav za potencialno aplikacijo v tehniki. Težava je bila pri pregledu literature ločiti dejstva, ki so lahko pomembna pri aplikacijah, od dejstev, ki so sicer zanimiva iz vidika naravoslovja, vendar se pri aplikacijah lahko zanemarijo.

4. Ocena stopnje realizacije zastavljenih raziskovalnih ciljev[4]

Namenski cilj raziskovalnega projekta je bil raziskava možnosti za uporabo nanocevk pri razvoju separacijskih membran v procesni tehniki, medtem ko je bil objektni cilj projekta razvoj modela transporta večkomponentne plinske mešanice skozi enostenske ogljikove nanocevke. Cilja projekta sta bila realizirana. Rezultati so potrdili raziskovalno hipotezo o selektivnosti ogljikovih nanocevk in njihovi potencialni uporabi za procese separacije.

5. Utemeljitev morebitnih sprememb programa raziskovalnega projekta oziroma sprememb, povečanja ali zmanjšanja sestave projektne skupine[5]

Pri programu raziskovalnega projekta ni prišlo do bistvenih sprememb.

6. Najpomembnejši znanstveni rezultati projektne skupine[6]

	Znanstveni rezultat		
1.	Naslov	*SLO*	X
		ANG	X
	Opis	*SLO*	Ker so na najpomembnejši letni konferenci na področju nanocevk (glej točko 7) objavljeni samo povzetki, prispevkov na tej konferenci ni mogoče uvrstiti v to rubriko. Ker v poročilih za predhodna leta v to rubriko niso spadali objavljeni referati na konferencah, je pri udeležbi na konferencah bila dana prednost kakovosti in nivoju konference pred možnostjo objave referata v zborniku. V primeru, da bi bili kriteriji veljavni od začetka projekta, bi bili rezultati dela predstavljeni tudi na konferencah, kjer so referati objavljeni v zborniku.
		ANG	X
	Objavljeno v		X
	Tipologija		1.08 Objavljeni znanstveni prispevek na konferenci
	COBISS.SI-ID		1
2.	Naslov	*SLO*	
		ANG	
	Opis	*SLO*	
		ANG	
	Objavljeno v		
	Tipologija		
	COBISS.SI-ID		
3.	Naslov	*SLO*	
		ANG	
	Opis	*SLO*	
		ANG	
	Objavljeno v		
	Tipologija		
	COBISS.SI-ID		
4.	Naslov	*SLO*	
		ANG	
	Opis	*SLO*	
		ANG	
	Objavljeno v		
	Tipologija		
	COBISS.SI-ID		

5.	Naslov	SLO	
		ANG	
	Opis	SLO	
		ANG	
	Objavljeno v		
	Tipologija		
	COBISS.SI-ID		

7. Najpomembnejši družbeno-ekonomsko relevantni rezultati projektne skupine[6]

	Družbeno-ekonomsko relevantni rezultat		
1.	Naslov	SLO	Velekanonična Monte-Carlo simulacija adsorpcije mešanic H2/N2 in H2/O2 v enostenskih ogljikovih nanocevkah
		ANG	Grand Canonical Monte-Carlo Simulation of Adsorption of H2/N2 and H2/O2 Mixtures in Single-Wall Carbon Nanotubes
	Opis	SLO	Izvedene so bile velekanonične Monte-Carlo simulacije adsorpcije mešanic H2/N2 in H2/O2 v enostenski ogljikovi nanocevki. Medsebojne interakcije so bile modelirane z uporabo Lennard-Jonesovega potenciala. Obravnavane so bile toge nanocevke petih različnih premerov od 1,08 nm do 5,42 nm. Predpostavljena je bila t.i. sobna temperature (25 ºC), medtem ko je tlak zavzemal vrednosti od 10 do 1000 bar. Prikazano je bilo obnašanje posameznih komponent mešanic znotraj nanocevke in analizirana adsorpcijska selektivnost.
		ANG	Grand Canonical Monte-Carlo simulations of H2/N2 and H2/O2 mixtures in single-wall carbon nanotubes were carried out. All interactions were modelled via the Lennard-Jones potential. Armchair nanotubes were considered, with diameters ranging from 1.08 up to 5.42 nm. So-called room temperature was assumed (25 ºC), whereas the pressure varied from 10 up to 1000 bars. The behaviour of individual mixture components (i.e., molecules) within the nanotubes and adsorption selectivities at different conditions was investigated.
	Šifra		B.03 Referat na mednarodni znanstveni konferenci
	Objavljeno v		KLJENAK, Ivo. Grand Canonical Monte-Carlo Simulation of Adsorption of H2/N2 and H2/O2 Mixtures in Single-Wall Carbon Nanotubes, Book of Abstracts, Ninth International Conference on the Science and Application of Nanotubes, Montpellier, France, 29.6-4.7.2008
	Tipologija		1.12 Objavljeni povzetek znanstvenega prispevka na konferenci
	COBISS.SI-ID		21904167
2.	Naslov	SLO	Simulacija adsorpcije in difuzije trokomponentne plinske mešanice O2/N2/CO2 v enostenskih ogljikovih nanocevkah
		ANG	Simulation of adsorption and diffusion of ternary O2/N2/CO2 gas mixture in single-wall carbon nanotubes
	Opis	SLO	Izvedene so bile simulacije adsorpcije in difuzije trokomponentne plinske mešanice O2/N2/CO2 v enostenskih ogljikovih nanocevkah. Obravnavane so bile toge nanocevke treh različnih premerov: 1,356 nm, 2,170 nm in 3,255 nm. Predpostavljena je bila temperatura 298K, medtem ko je tlak zavzemal vrednosti od 1 do 100 bar. Prikazano je bilo obnašanje posameznih komponent mešanice znotraj nanocevk. Prav tako je bila določena samodifuzivnost plina v najožji nanocevki.
		ANG	Simulations of adsorption and diffusion of a ternary O2/N2/CO2 gas mixture in single-wall carbon nanotubes were performed. Rigid nanotubes of three different diameters were considered: 1.356 nm, 2.170 nm and 3.255 nm. A temperature of 298K was assumed, whereas the pressure assumed values from 1 to 100 bars. The behaviour of individual mixture components whitin the nanotubes was presented. The gas self-diffusivity in the narrowest nanotube was also determined.
	Šifra		B.03 Referat na mednarodni znanstveni konferenci
	Objavljeno v		KLJENAK, Ivo. Simulation of adsorption and diffusion of ternary O2/N2/CO2 gas mixture in single-wall carbon nanotubes, 10th International Conference

			on the Science and Application of Nanotubes, June 21-26, 2009, Beijing, China. NT09 Book of Abstracts, p. 295.
	Tipologija		1.12 Objavljeni povzetek znanstvenega prispevka na konferenci
	COBISS.SI-ID		22735143
3.	Naslov	*SLO*	Simulacija ternarne mešanice O2/N2/CO2 v enostenski ogljikovi nanocevki z metodo molekularne dinamike
		ANG	Molecular dynamics simulation of ternary O2/N2/CO2 mixture flow in single-wall carbon nanotubes
	Opis	*SLO*	Z metodo neravnovesne molekularne dinamike so bile izvedene simulacije ternarne plinske mešanice O2/N2/CO2 skozi enostenske odprte ogljikove nanocevke. Interakcije so bile modelirane z Lennard-Jonesovim potencialom. Ravnovesne konfiguracije so bile dobljene z velekanonično Monte-Carlo simulacijo. Proces uravnovešenja je implicitno simuliral adsorpcijo plina na steni nanocevke. Tok mešanice je bil nato sprožen z ustvarjanjem višje molekularno gostoto na enem koncu cevke. Obravnavane so bile nanocevke z naslednjimi premeri: 1.356 nm, 2.170 nm in 3.255 nm.
		ANG	Non-equilibrium molecular dynamics simulations of the flow of an O2/N2/CO2 mixture through single-wall open-ended carbon nanotubes were performed. Interactions were modelled via the Lennard-Jones potential. Equilibrium configurations were established in the nanotubes by performing Grand Canonical Monte-Carlo simulations. The equilibration process simulated gas adsorption on the nanotube wall. The flow of the mixture was started by creating a higher molecular density at one tube end. Nanotubes with the folowing diameters were considered: 1.356 nm, 2.170 nm and 3.255 nm.
	Šifra		B.03 Referat na mednarodni znanstveni konferenci
	Objavljeno v		KLJENAK, Ivo. Molecular dynamics simulation of ternary O2/N2/CO2 mixture flow in single-wall carbon nanotubes. 11th International Conference on the Science and Application of Nanotubes, June 27-July 2, 2010, Montréal, Québec, Canada. NT10 Book of Abstracts, p.140.
	Tipologija		1.12 Objavljeni povzetek znanstvenega prispevka na konferenci
	COBISS.SI-ID		23791399
4.	Naslov	*SLO*	
		ANG	
	Opis	*SLO*	
		ANG	
	Šifra		
	Objavljeno v		
	Tipologija		
	COBISS.SI-ID		
5.	Naslov	*SLO*	
		ANG	
	Opis	*SLO*	
		ANG	
	Šifra		
	Objavljeno v		
	Tipologija		
	COBISS.SI-ID		

8. Drugi pomembni rezultati projetne skupine[8]

9. Pomen raziskovalnih rezultatov projektne skupine[9]

9.1. Pomen za razvoj znanosti[10]

SLO

Raziskave spadajo v splošno področje raziskav pojavov na nivoju nanoskale. Raziskave omogočajo nova dognanja o lastnostih in obnašanju tekočine, obdane na nanoskali:
— adsorpcijska selektivnost ogljikovih nanocevk,
— lastnosti difuzije plinov v nanocevkah,
— lastnosti difuzije posameznih komponent v večkomponentnih zmeseh plinov v nanocevkah,
— jakost konvektivnega pretoka plina skozi nanocevke.

ANG

The research belongs to the general field of research of phenomena on the nanoscale. The research generates new knowledge on the properties and behaviour on fluids, confined at the nanoscale:
— adsorption selectivity of carbon nanotubes,
— properties of gas diffusion in nanotubes,
— properties of diffusion of particular components in multicomponent gas mixtures in nanotubes,
— convective flowrate of gases through nanotubes.

9.2. Pomen za razvoj Slovenije[11]

SLO

Aplikacija nanocevk v tehniki je doslej zajemala predvsem uporabo v elektroniki (zaradi njihovih prevodniških lastnosti) in materialih (zaradi njihovih trdnostnih lastnosti), medtem ko je uporaba v procesnem strojništvu bila manj poudarjena. Nanocevke bo mogoče izkoristiti v procesnem strojništvu zaradi specifičnih lastnosti prenosa (toka zaradi razlike tlakov in difuzije) tekočin (plinov in kapljevin) na nanoskali. Rezultati predlaganih raziskav bi v začetni fazi lahko bili aplicirani na uporabo nanocevk pri razvoju separacijskih membran za industrijske postopke v procesni tehniki ali pa na drugih področjih procesnega strojništva. Dosedanje teoretične raziskave namreč kažejo, da bi lahko membrane na osnovi nanocevk imele hkrati visoki selektivnost in pretok, kar je že dolgo cilj tehnologije membran. Simulacije toka tekočin so lahko pozneje uporabne tudi pri načrtovanju drugih sistemov, v katerih se prenos tekočin vrši v nanocevkah.

ANG

So far, nanotubes have been applied in technical applications mostly in electronics (because of their electrical conductivity properties) and materials (because of their structural properties), whereas their use in process engineeering was less emphasized. Nanotubes could be used in process engineering due to specific properties of fluid (liquid and gas) transport (pressure-driven flow and diffusion) on the nanoscale. In the initial phase, the results of the proposed research could be applied to the use of nanotubes in the development of separation membranes for industrial processes in unit operations or other fields in process engineering. Namely, theoretical investigations show that nanotube-based membranes could have both high selectivity and fluxes, which is a long standing goal of membrane technology. Simulations of fluid flow could later be used in the design of other systems, where fluid is being transported through nanotubes.

10. Samo za aplikativne projekte!
Označite, katerega od navedenih ciljev ste si zastavili pri aplikativnem projektu, katere konkretne rezultate ste dosegli in v kakšni meri so doseženi rezultati uporabljeni

Cilj		
F.01	**Pridobitev novih praktičnih znanj, informacij in veščin**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.02	**Pridobitev novih znanstvenih spoznanj**	
	Zastavljen cilj	○ DA ○ NE

	Rezultat	
	Uporaba rezultatov	
F.03	**Večja usposobljenost raziskovalno-razvojnega osebja**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.04	**Dvig tehnološke ravni**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.05	**Sposobnost za začetek novega tehnološkega razvoja**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.06	**Razvoj novega izdelka**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.07	**Izboljšanje obstoječega izdelka**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.08	**Razvoj in izdelava prototipa**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.09	**Razvoj novega tehnološkega procesa oz. tehnologije**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.10	**Izboljšanje obstoječega tehnološkega procesa oz. tehnologije**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.11	**Razvoj nove storitve**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	

F.12	**Izboljšanje obstoječe storitve**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.13	**Razvoj novih proizvodnih metod in instrumentov oz. proizvodnih procesov**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.14	**Izboljšanje obstoječih proizvodnih metod in instrumentov oz. proizvodnih procesov**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.15	**Razvoj novega informacijskega sistema/podatkovnih baz**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.16	**Izboljšanje obstoječega informacijskega sistema/podatkovnih baz**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.17	**Prenos obstoječih tehnologij, znanj, metod in postopkov v prakso**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.18	**Posredovanje novih znanj neposrednim uporabnikom (seminarji, forumi, konference)**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.19	**Znanje, ki vodi k ustanovitvi novega podjetja ("spin off")**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.20	**Ustanovitev novega podjetja ("spin off")**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.21	**Razvoj novih zdravstvenih/diagnostičnih metod/postopkov**	

	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.22	**Izboljšanje obstoječih zdravstvenih/diagnostičnih metod/postopkov**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.23	**Razvoj novih sistemskih, normativnih, programskih in metodoloških rešitev**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.24	**Izboljšanje obstoječih sistemskih, normativnih, programskih in metodoloških rešitev**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.25	**Razvoj novih organizacijskih in upravljavskih rešitev**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.26	**Izboljšanje obstoječih organizacijskih in upravljavskih rešitev**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.27	**Prispevek k ohranjanju/varovanje naravne in kulturne dediščine**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.28	**Priprava/organizacija razstave**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.29	**Prispevek k razvoju nacionalne kulturne identitete**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	
	Uporaba rezultatov	
F.30	**Strokovna ocena stanja**	
	Zastavljen cilj	○ DA ○ NE
	Rezultat	

	Uporaba rezultatov	
F.31	**Razvoj standardov**	
	Zastavljen cilj	◯ DA ◯ NE
	Rezultat	
	Uporaba rezultatov	
F.32	**Mednarodni patent**	
	Zastavljen cilj	◯ DA ◯ NE
	Rezultat	
	Uporaba rezultatov	
F.33	**Patent v Sloveniji**	
	Zastavljen cilj	◯ DA ◯ NE
	Rezultat	
	Uporaba rezultatov	
F.34	**Svetovalna dejavnost**	
	Zastavljen cilj	◯ DA ◯ NE
	Rezultat	
	Uporaba rezultatov	
F.35	**Drugo**	
	Zastavljen cilj	◯ DA ◯ NE
	Rezultat	
	Uporaba rezultatov	

Komentar

11. Samo za aplikativne projekte!
Označite potencialne vplive oziroma učinke vaših rezultatov na navedena področja

	Vpliv	**Ni vpliva**	**Majhen vpliv**	**Srednji vpliv**	**Velik vpliv**	
G.01	**Razvoj visoko-šolskega izobraževanja**					
G.01.01.	Razvoj dodiplomskega izobraževanja	◯	◯	◯	◯	
G.01.02.	Razvoj podiplomskega izobraževanja	◯	◯	◯	◯	
G.01.03.	Drugo:	◯	◯	◯	◯	
G.02	**Gospodarski razvoj**					
G.02.01	Razširitev ponudbe novih izdelkov/storitev na trgu	◯	◯	◯	◯	
G.02.02.	Širitev obstoječih trgov	◯	◯	◯	◯	
G.02.03.	Znižanje stroškov proizvodnje	◯	◯	◯	◯	
G.02.04.	Zmanjšanje porabe materialov in energije	◯	◯	◯	◯	
G.02.05.	Razširitev področja dejavnosti	◯	◯	◯	◯	

G.02.06.	Večja konkurenčna sposobnost	○	○	○	○	
G.02.07.	Večji delež izvoza	○	○	○	○	
G.02.08.	Povečanje dobička	○	○	○	○	
G.02.09.	Nova delovna mesta	○	○	○	○	
G.02.10.	Dvig izobrazbene strukture zaposlenih	○	○	○	○	
G.02.11.	Nov investicijski zagon	○	○	○	○	
G.02.12.	Drugo:	○	○	○	○	
G.03	**Tehnološki razvoj**					
G.03.01.	Tehnološka razširitev/posodobitev dejavnosti	○	○	○	○	
G.03.02.	Tehnološko prestrukturiranje dejavnosti	○	○	○	○	
G.03.03.	Uvajanje novih tehnologij	○	○	○	○	
G.03.04.	Drugo:	○	○	○	○	
G.04	**Družbeni razvoj**					
G.04.01	Dvig kvalitete življenja	○	○	○	○	
G.04.02.	Izboljšanje vodenja in upravljanja	○	○	○	○	
G.04.03.	Izboljšanje delovanja administracije in javne uprave	○	○	○	○	
G.04.04.	Razvoj socialnih dejavnosti	○	○	○	○	
G.04.05.	Razvoj civilne družbe	○	○	○	○	
G.04.06.	Drugo:	○	○	○	○	
G.05.	**Ohranjanje in razvoj nacionalne naravne in kulturne dediščine in identitete**	○	○	○	○	
G.06.	**Varovanje okolja in trajnostni razvoj**	○	○	○	○	
G.07	**Razvoj družbene infrastrukture**					
G.07.01.	Informacijsko-komunikacijska infrastruktura	○	○	○	○	
G.07.02.	Prometna infrastruktura	○	○	○	○	
G.07.03.	Energetska infrastruktura	○	○	○	○	
G.07.04.	Drugo:	○	○	○	○	
G.08.	**Varovanje zdravja in razvoj zdravstvenega varstva**	○	○	○	○	
G.09.	**Drugo:**	○	○	○	○	

Komentar

12. Pomen raziskovanja za sofinancerje, navedene v 2. točki [12]

1.	**Sofinancer**		
	Vrednost sofinanciranja za celotno obdobje		**EUR**

	trajanja projekta je znašala:			
	Odstotek od utemeljenih stroškov projekta:			%
	Najpomembnejši rezultati raziskovanja za sofinancerja			**Šifra**
		1.		
		2.		
		3.		
		4.		
		5.		
	Komentar			
	Ocena			
2.	**Sofinancer**			
	Vrednost sofinanciranja za celotno obdobje trajanja projekta je znašala:			EUR
	Odstotek od utemeljenih stroškov projekta:			%
	Najpomembnejši rezultati raziskovanja za sofinancerja			**Šifra**
		1.		
		2.		
		3.		
		4.		
		5.		
	Komentar			
	Ocena			
3.	**Sofinancer**			
	Vrednost sofinanciranja za celotno obdobje trajanja projekta je znašala:			EUR
	Odstotek od utemeljenih stroškov projekta:			%
	Najpomembnejši rezultati raziskovanja za sofinancerja			**Šifra**
		1.		
		2.		
		3.		
		4.		
		5.		

	Komentar				
	Ocena				

C. IZJAVE

Podpisani izjavljam/o, da:

- so vsi podatki, ki jih navajamo v poročilu, resnični in točni
- se strinjamo z obdelavo podatkov v skladu z zakonodajo o varstvu osebnih podatkov za potrebe ocenjevanja, za objavo 6., 7. in 8. točke na spletni strani http://sicris.izum.si/ ter obdelavo teh podatkov za evidence ARRS
- so vsi podatki v obrazcu v elektronski obliki identični podatkom v obrazcu v pisni obliki
- so z vsebino zaključnega poročila seznanjeni in se strinjajo vsi soizvajalci projekta

Podpisi:

Ivo Kljenak	in	
podpis vodje raziskovalnega projekta		zastopnik oz. pooblaščena oseba RO

Kraj in datum:	Ljubljana	21.4.2011

Oznaka poročila: ARRS-RPROJ-ZP-2011-1/162

---

[1] Zaradi spremembe klasifikacije družbeno ekonomskih ciljev je potrebno v poročilu opredeliti družbeno ekonomski cilj po novi klasifikaciji. Nazaj

[2] Samo za aplikativne projekte. Nazaj

[3] Napišite kratko vsebinsko poročilo, kjer boste predstavili raziskovalno hipotezo in opis raziskovanja. Navedite ključne ugotovitve, znanstvena spoznanja ter rezultate in učinke raziskovalnega projekta. Največ 18.000 znakov vključno s presledki (približno tri strani, velikosti pisave 11). Nazaj

[4] Realizacija raziskovalne hipoteze. Največ 3.000 znakov vključno s presledki (približno pol strani, velikosti pisave 11). Nazaj

[5] V primeru bistvenih odstopanj in sprememb od predvidenega programa raziskovalnega projekta, kot je bil zapisan v predlogu raziskovalnega projekta oziroma v primeru sprememb, povečanja ali zmanjšanja sestave projektne skupine v zadnjem letu izvajanja projekta (obrazložitev). V primeru, da sprememb ni bilo, to navedite. Največ 6.000 znakov vključno s presledki (približno ena stran, velikosti pisave 11). Nazaj

[6] Navedite največ pet najpomembnejših znanstvenih rezultatov projektne skupine, ki so nastali v času trajanja projekta v okviru raziskovalnega projekta, ki je predmet poročanja. Za vsak rezultat navedite naslov v slovenskem in angleškem jeziku (največ 150 znakov vključno s presledki), rezultat opišite (največ 600 znakov vključno s presledki) v slovenskem in angleškem jeziku, navedite, kje je objavljen (največ 500 znakov vključno s presledki), izberite ustrezno šifro tipa objave po Tipologiji dokumentov/del za vodenje bibliografij v sistemu COBISS ter napišite ustrezno COBISS.SI-ID številko bibliografske enote.
Navedeni rezultati bodo objavljeni na spletni strani http://sicris.izum.si/.

PRIMER (v slovenskem jeziku):
Naslov: Regulacija delovanja beta-2 integrinskih receptorjev s katepsinom X;
Opis: Cisteinske proteaze imajo pomembno vlogo pri nastanku in napredovanju raka. Zadnje študije kažejo njihovo povezanost s procesi celičnega signaliziranja in imunskega odziva. V tem znanstvenem članku smo prvi dokazali... (največ 600 znakov vključno s presledki)
Objavljeno v: OBERMAJER, N., PREMZL, A., ZAVAŠNIK-BERGANT, T., TURK, B., KOS, J.. Carboxypeptidase cathepsin X mediates ß2 - integrin dependent adhesion of differentiated U-937 cells. Exp. Cell Res., 2006, 312, 2515-2527, JCR IF (2005): 4.148
Tipopologija: 1.01 - Izvirni znanstveni članek
COBISS.SI-ID: 1920113 Nazaj

[7] Navedite največ pet najpomembnejših družbeno-ekonomsko relevantnih rezultatov projektne skupine, ki so nastali v času trajanja projekta v okviru raziskovalnega projekta, ki je predmet poročanja. Za vsak rezultat navedite naslov (največ 150 znakov vključno s presledki), rezultat opišite (največ 600 znakov vključno s presledki), izberite ustrezen

rezultat, ki je v Šifrantu raziskovalnih rezultatov in učinkov (Glej: http://www.arrs.gov.si/sl/gradivo/sifranti/sif-razisk-rezult.asp), navedite, kje je rezultat objavljen (največ 500 znakov vključno s presledki), izberite ustrezno šifro tipa objave po Tipologiji dokumentov/del za vodenje bibliografij v sistemu COBISS ter napišite ustrezno COBISS.SI-ID številko bibliografske enote.
Navedeni rezultati bodo objavljeni na spletni strani http://sicris.izum.si/. Nazaj

[8] Navedite rezultate raziskovalnega projekta v primeru, da katerega od rezultatov ni mogoče navesti v točkah 6 in 7 (npr. ker se ga v sistemu COBISS ne vodi). Največ 2.000 znakov vključno s presledki. Nazaj

[9] Pomen raziskovalnih rezultatov za razvoj znanosti in za razvoj Slovenije bo objavljen na spletni strani: http://sicris.izum.si/ za posamezen projekt, ki je predmet poročanja. Nazaj

[10] Največ 4.000 znakov vključno s presledki Nazaj

[11] Največ 4.000 znakov vključno s presledki Nazaj

[12] Rubrike izpolnite/prepišite skladno z obrazcem "Izjava sofinancerja" (http://www.arrs.gov.si/sl/progproj/rproj/gradivo/), ki ga mora izpolniti sofinancer. Podpisan obrazec "Izjava sofinancerja" pridobi in hrani nosilna raziskovalna organizacija – izvajalka projekta. Nazaj

Obrazec: ARRS-RPROJ-ZP/2011-1 v1.01
A6-CD-96-48-CB-D6-6F-B7-A0-C4-C6-44-B7-3A-18-1D-46-D2-24-96

```
/***   NANOFLOW.C   ***/

/* Ivo Kljenak            */
/* Jozef Stefan Institute */
/* Ljubljana, Slovenia    */

/* Set for simulation:
1. TUBE_DIAMETER, TUBE_HEIGHT
2. COMP1, COMP2, COMP3, RATIO_COMP1, RATIO_COMP2, RATIO_COMP3
3. PRESSURE_ACTUAL
4. TEMPERATURE_ACTUAL
5. File tubexyz.txt
*/

/* For MOLECULAR_DYNAMICS:
   Create file of carbon nanotube from basic unit cell
   Eventually adjust TUBE_HEIGHT
   Eventually modify initial coordinates of gas molecules
*/

#include  <io.h>
#include  <math.h>
#include  <conio.h>
#include  <fcntl.h>
#include  <sys\stat.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <string.h>
#include  "mshell_2.h"
#include  "general.h"

#define MOLECULAR_DYNAMICS /* GRAND_CANONICAL CANONICAL */
/* #define WIDOM           */
/* #define MINIMUM_IMAGE   */
#define NPART_MAX  3000

#if defined (GRAND_CANONICAL)
  #define NPART_INITIAL  200
#endif

#define  NANOTUBE
#define  CONFINED_PARTICLES

/* #define  CHECK_PARTICLES_ORDER */
/* #define  CHECK_ENERGY          */

/* Physical constants */
#define  BOLTZMANN       1.3806504e-23  /* J/K    */
#define  R_GAS           8.314          /* J/molK */
#define  PLANCK          6.626068e-34
#define  AVOGADRO        6.022e+23

/* Adjustable parameters */
#define  R_MINIMUM        0.32e-10
#define  CUTOFF_DISTANCE 100.0e-10
/* minimum distance between atoms: 64e-12 m */

#define  COMP1  'N'    /* H:H2 N:N2 O:O2 C:CO2 M:CH4 */
#define  COMP2  'O'
```

```
#define  COMP3  'C'

/* Mixture composition and physical conditions */
#if defined (GRAND_CANONICAL) || defined (CANONICAL)
  #define  RATIO_COMP1   1.0  /* 0.395  /* 0.79/2 */
  #define  RATIO_COMP2   0.0  /* 0.105  /* 0.21/2 */
  #define  PRESSURE_ACTUAL    1.0e+05
#endif
#define  TEMPERATURE_ACTUAL  298.0

/* Geometry */
/* TUBE_HEIGHT is the actual height of the nanotube, that will be used */
/* When running MOLECULAR_DYNAMICS, it is not necessarily the same value,
   that was used during the GRAND_CANONICAL simulation
   If a larger value is used, then the initial coordinates of the gas
   molecules are adjusted before the start of the MOLECULAR_DYNAMICS
   simulation
*/
#define  LATTICE_CYLIND /* LATTICE_SQUARE */
#if defined (LATTICE_CYLIND)
  #define  TUBE_DIAMETER        21.70e-10
  #define  TUBE_HEIGHT      295.14146e-10
#elif defined (LATTICE_SQUARE)
  #define  BOX_SQUARE       3.0e-8
#endif

/* Tube diameters and lengths (40 unit cells) */
/* (8,8):    10.85  98.38049 */
/* (10,10):  13.56  97.15073 */
/* (16,16):  21.70  98.38049 */
/* (24,24):  32.55  98.38049 */
/* (32,32):  43.40  98.38049 */
/* (40,40):  54.25  97.15073 */

/* Simulation parameters */
#if defined (GRAND_CANONICAL) || defined (CANONICAL)
  #define  DR_INIT         0.2e-10
  #define  EQUIL_CYCLES    20000
  #define  PROD_CYCLES     20000
  #define  DISPLACEM       200
  #if defined (GRAND_CANONICAL)
    #define  EXCHANGES     800
  #endif
#elif defined (MOLECULAR_DYNAMICS)
  #define  DT              0.2e-15   /* Lee & Sinnott: 0.2 fs = 0.2e-15 s */
  #define  TIME_EQUIL      1.0e-12   /* Cannon: 50 ps = 50e-12 s          */
  #define  TIME_AVER      10.0e-12   /* Cannon: 20 ns = 20e-9 s           */
  #define  DT_AVER_DIFFUS  1.0e-13
#endif

/* #define TAIL_CORRECT */
/* #define SHIFT_POTENT */
#define WRITE_CONFIG

/* N2 - 'N' */
#define  MOLECULAR_MASS_N2   28.02e-3
#define  SIGMA_N2            3.32e-10
#define  EPSILON_OVER_KB_N2  36.4
```

```
/* H2 - 'H' */
#define  MOLECULAR_MASS_H2   2.016e-3
#define  SIGMA_H2            2.958e-10
#define  EPSILON_OVER_KB_H2  36.7

/* O2 - 'O' */
#define  MOLECULAR_MASS_O2   32.00e-3
#define  SIGMA_O2            2.99e-10
#define  EPSILON_OVER_KB_O2  52.0

/* CO2 - 'C' */
#define  MOLECULAR_MASS_CO2   44.00e-3
#define  SIGMA_CO2            3.454e-10
#define  EPSILON_OVER_KB_CO2  235.90

/* CH4 - 'M' */
#define  MOLECULAR_MASS_CH4   16.032e-3
#define  SIGMA_CH4            3.734e-10
#define  EPSILON_OVER_KB_CH4  147.90

/* C-C */
#define  SIGMA_CC             3.55e-10
#define  EPSILON_OVER_KB_CC   35.26

/***  global variables   ***/
/*  temp: temperature          */
/*  beta: 1/(kb*temp)          */
/*  grand canonical coupled to an ideal gas bath with pressure: pressure_ideal */
/*  chemical potential bath: mu^b = mu^0 + ln(beta*pressure_ideal)/beta        */
/*                                = ln (beta*pressure_ideal*Lamda^3)/beta      */
/*  zz is defined as           zz = exp(beta*mu^b)/Lamda^3                     */
/*                                = beta*pressure_ideal                        */
/*  excess chemical potent.  muex = mu^b -mu^id                                */
/*                      = mu^0 + ln(beta*pressure_ideal)/beta - mu^0 - ln(rho) */
/*                                  = ln(zz)/beta - ln <rho>                   */

double  *x, *y, *z;                  /* x(i),y(i),z(i): position of particle i */
int     *comp;                       /* 1 or 2  or 3                           */
int     npart, npart_comp [4];       /* actual number of particles             */

#if defined (MOLECULAR_DYNAMICS)
  double  *mass;
  double  time, dt, dt2, two_dt, t_max;
  double  *x_prev, *y_prev, *z_prev, *z_initial;
  double  *x_prev_prev, *y_prev_prev, *z_prev_prev;
  double  *v_x, *v_y, *v_z;
  double  *f_x, *f_y, *f_z;
  double  mass_particle_1, mass_particle_2;
  #if defined (COMP3)
    double  mass_particle_3;
  #endif
#endif

#if defined (GRAND_CANONICAL) || defined (CANONICAL)
  double  rho_mass, rho_particles_1, rho_particles_2;
  double  mu_1, mu_2;             /* chemical potential per particle  */
  double  lambda_1, lambda_2;     /* de Broglie thermal length        */
  double  beta, zz_1, zz_2;
  #if defined (COMP3)
```

```
    double  zz_3, mu_3, lambda_3, rho_particles_3;
  #endif
#endif

double  temp;
/* zz = exp(beta*mu^b)/Lamda^3: related to chemical potential reservoir */
double  mass_molecular_1, mass_molecular_2;
#if defined (COMP3)
  double  mass_molecular_3;
#endif

#if defined (GRAND_CANONICAL) || defined (WIDOM)
  double  pressure;
  double  mu_ideal_gas;
#endif

/***  geometrical parameters   ***/
#if defined (LATTICE_CYLIND)
  double  tube_height, tube_radius, tube_radius_2, tube_volume;
#elif defined (LATTICE_SQUARE)
  /* box: simulation box length;   half_box: 0.5 * box */
  double  box_side, box_half, tube_volume;
#endif

#if defined (NANOTUBE)
  int     n_carbon;
  double  *x_carbon, *y_carbon, *z_carbon;
#endif

/* eps4           :4 * epsilon
   eps48          :48 * epsilon
   (epsilon)      :energy parameter Lennard-Jones potential
   sigma2         :sigma * sigma
   sigma          :size parameter Lennard-Jones potential
   r_cutoff       :cut-off radius of the potential for particles
   r_cutoff_wall  :cut-off vertical distance for particles and wall
   r_cutoff_2     :r_cutoff * r_cutoff
   r_minimum      :minimum distance between atoms
   r_minimum_2    :r_minimum * r_minimum
   energy_cutoff  :energy at cut-off radius
*/

double  eps4_1_1,  eps4_2_2,   eps4_1_2,  eps4_C_1,  eps4_C_2;
double  eps48_1_1, eps48_2_2,  eps48_1_2, eps48_C_1, eps48_C_2;
double  sigma_1_1, sigma2_1_1, sigma3_1_1;
double  sigma_2_2, sigma2_2_2, sigma3_2_2;
double  sigma_1_2, sigma2_1_2, sigma3_1_2;
#if defined (COMP3)
  double  eps4_3_3,  eps4_1_3,   eps4_2_3,  eps4_C_3;
  double  eps48_3_3, eps48_1_3,  eps48_2_3, eps48_C_3;
  double  sigma_3_3, sigma2_3_3, sigma3_3_3;
  double  sigma_1_3, sigma2_1_3, sigma3_1_3;
  double  sigma_2_3, sigma2_2_3, sigma3_2_3;
#endif

#if defined (NANOTUBE)
  double  sigma_C_1, sigma2_C_1, sigma3_C_1;
  double  sigma_C_2, sigma2_C_2, sigma3_C_2;
  #if defined (COMP3)
```

```
    double sigma_C_3, sigma2_C_3, sigma3_C_3;
  #endif
#endif
double  energy_cutoff;
double  r_cutoff,      r_cutoff_2;
double  r_minimum,     r_minimum_2;
double  r_cutoff_wall, r_cutoff_wall_2;


/* uniform random number generation between 0.0 and 1.0 */
double random1000 (void)
{
  double rnd1000;

  rnd1000 = (double) random (1000);
  rnd1000 /= 1000;

  return (rnd1000);
}

#if !defined (MOLECULAR_DYNAMICS)

/***  tail correction: pressure  ***/
double correct_p (double R, double rho_particles, double sigma3, double eps4)
{
  double ri3;
  double pressure_correct;

  if (R < 1.0E-36)
    {
      printf ("R small in correct_p\n");
      printf ("Press any key ...");
      getch ();
      printf ("\n");
    }
  ri3  = sigma3 / (R * R * R);
  pressure_correct = 4.0 * M_PI * eps4 * (rho_particles * rho_particles)
     * sigma3 * (2.0 * ri3 * ri3 * ri3 / 9.0 - ri3 / 3.0);
  return (pressure_correct);
}


/***  tail correction: energy    ***/
double correct_u (double R, double rho_particles, double sigma3, double eps4)
{
  double ri3;

  if (R < 1.0E-36)
    {
      printf ("R small in correct_u\n");
      printf ("Press any key ...");
      getch ();
      printf ("\n");
    }
  ri3  = sigma3 / (R * R * R);
  return (2.0 * M_PI * eps4 * (rho_particles * sigma3)
    * (ri3 * ri3 * ri3 / 9.0 - ri3 / 3.0));
}
```

```
#endif /* !MOLECULAR_DYNAMICS */

/* calculates energy (en) and virial (vir) for given squared distance between two
particles */
void energy_function (double *En, double *Vir, double r2, double sigma2, double
eps4, double eps48, double r_cutoff_2)
{
  double r2i, r6i;

  if (r2 <= r_cutoff_2)
    {
      r2i = sigma2 / r2;
      r6i = r2i * r2i * r2i;
      *En = eps4 * (r6i * r6i - r6i);
      #if defined (SHIFT_POTENT)
        *En -= energy_cutoff;
      #endif

      *Vir = eps48 * (r6i * r6i - 0.5 * r6i);
    }
  else
    {
      *En  = 0.0;
      *Vir = 0.0;
    }
  return;
}


/* calculates force for given distance between two particles  */
/* the force is positive (repulsive) or negative (attractive) */
double force_generic_function (double r2, double sigma2, double eps48)
{
  double r2i, r6i;
  double force_generic;

  if (r2 <= r_cutoff_2)
    {
      r2i = sigma2 / r2;
      r6i = r2i * r2i * r2i;
      force_generic = eps48 * (r6i * r6i - 0.5 * r6i) / r2;
    }
  else
    {
      force_generic = 0.0;
    }
  return (force_generic);
}


void energy_particle_function (double Xi, double Yi, double Zi, int I, int j_begin,
double *En, double *Vir)
{
  double dx, dy, dz, r2, virial_ij, energy_ij;
  int    j;
  *En  = 0.0;
  *Vir = 0.0;

  for (j = j_begin; j < npart; j++)
```

```
{
  if (j != I)
    {
      dx = Xi - x [j];
      dy = Yi - y [j];
      dz = Zi - z [j];

      /* minimum image */
      #if defined (MINIMUM_IMAGE)
        #if defined (LATTICE_CYLIND)
          if (dx > tube_radius)
            dx -= 2.0*tube_radius;
          else if (dx < -tube_radius)
            dx += 2.0*tube_radius;

          if (dy > tube_radius)
            dy -= 2.0*tube_radius;
          else if (dy < -tube_radius)
            dy += 2.0*tube_radius;

          if (dz > (tube_height/2))
            dz -= tube_height;
          else if (dz < -(tube_height/2))
            dz += tube_height;
        #elif defined (LATTICE_SQUARE)
          if (dx > box_half)
            dx -= box_side;
        else if (dx < -box_half)
            dx += box_side;

        if (dy > box_half)
            dy -= box_side;
        else if (dy < -box_half)
            dy += box_side;

        if (dz > box_half)
            dz -= box_side;
        else if (dz < -box_half)
            dz += box_side;
        #endif
      #endif

      r2 = dx*dx + dy*dy + dz*dz;
      if (r2 < r_minimum_2/2)
        {
          printf ("r2 small in energy_particle_function\n");
          printf ("I:%5d  X:%14.5e  Y:%14.5e  Z:%14.5e\n", I, Xi, Yi, Zi);
          printf ("Press any key ...");
          getch  ();
          printf ("\n");
        }

      /* calculate energy and virial pair i,j */
      if (comp [I] == 1)
        {
          if (comp [j] == 1)
            energy_function (&energy_ij, &virial_ij, r2, sigma2_1_1, eps4_1_1,
        eps48_1_1, r_cutoff_2);
          else if (comp [j] == 2)
```

```
              energy_function (&energy_ij, &virial_ij, r2, sigma2_1_2, eps4_1_2,
          eps48_1_2, r_cutoff_2);
            #if defined (COMP3)
            else if (comp [j] == 3)
              energy_function (&energy_ij, &virial_ij, r2, sigma2_1_3, eps4_1_3,
          eps48_1_3, r_cutoff_2);
            #endif
            else
              exit (1);
          }
        else if (comp [I] == 2)
          {
            if (comp [j] == 1)
              energy_function (&energy_ij, &virial_ij, r2, sigma2_1_2, eps4_1_2,
          eps48_1_2, r_cutoff_2);
            else if (comp [j] == 2)
              energy_function (&energy_ij, &virial_ij, r2, sigma2_2_2, eps4_2_2,
          eps48_2_2, r_cutoff_2);
            #if defined (COMP3)
            else if (comp [j] == 3)
              energy_function (&energy_ij, &virial_ij, r2, sigma2_2_3, eps4_2_3,
          eps48_2_3, r_cutoff_2);
            #endif
            else
              exit (1);
          }
        #if defined (COMP3)
        else if (comp [I] == 3)
          {
            if (comp [j] == 1)
              energy_function (&energy_ij, &virial_ij, r2, sigma2_1_3, eps4_1_3,
          eps48_1_3, r_cutoff_2);
            else if (comp [j] == 2)
              energy_function (&energy_ij, &virial_ij, r2, sigma2_2_3, eps4_2_3,
          eps48_2_3, r_cutoff_2);
            else if (comp [j] == 3)
              energy_function (&energy_ij, &virial_ij, r2, sigma2_3_3, eps4_3_3,
          eps48_3_3, r_cutoff_2);
            else
              exit (1);
          }
        #endif
        *En  += energy_ij;
        *Vir += virial_ij;
      }
    }
  return;
}


double force_generic_function_2 (int i, int j, double r2)
{
  double force_generic;

  if (comp [i] == 1)
    {
      if (comp [j] == 1)
        force_generic = force_generic_function (r2, sigma2_1_1, eps48_1_1);
      else if (comp [j] == 2)
```

```
        force_generic = force_generic_function (r2, sigma2_1_2, eps48_1_2);
      #if defined (COMP3)
      else if (comp [j] == 3)
        force_generic = force_generic_function (r2, sigma2_1_3, eps48_1_3);
      #endif
      else
        exit (1);
    }
  else if (comp [i] == 2)
    {
      if (comp [j] == 1)
        force_generic = force_generic_function (r2, sigma2_1_2, eps48_1_2);
      else if (comp [j] == 2)
        force_generic = force_generic_function (r2, sigma2_2_2, eps48_2_2);
      #if defined (COMP3)
      else if (comp [j] == 3)
        force_generic = force_generic_function (r2, sigma2_2_3, eps48_2_3);
      #endif
      else
        exit (1);
    }
  #if defined (COMP3)
  else if (comp [i] == 3)
    {
      if (comp [j] == 1)
        force_generic = force_generic_function (r2, sigma2_1_3, eps48_1_3);
      else if (comp [j] == 2)
        force_generic = force_generic_function (r2, sigma2_2_3, eps48_2_3);
      else if (comp [j] == 3)
        force_generic = force_generic_function (r2, sigma2_3_3, eps48_3_3);
      else
        exit (1);
    }
  #endif
  return (force_generic);
}


/*** x,y,z component of force between particles i and j                     ***/
/* the sign of the force may be changed by the sign of dx, dy, dz             */
/* example: if i is above j, and the force is attractive (negative), force (i,j) */
/* < 0, so the function force (i,j) returns the force exerted by j on i       */
void force_particles_function (int i, int j, double *force_x, double *force_y,
double *force_z, double r_cutoff_2)
{
  double dx, dy, dz, r2;
  double force_generic;
  double z_virtual;

  dx = x [i] - x [j];
  dy = y [i] - y [j];
  dz = z [i] - z [j];

  *force_x = 0.0;
  *force_y = 0.0;
  *force_z = 0.0;

  r2 = dx*dx + dy*dy + dz*dz;
  if (r2 < r_minimum_2/2)
```

```
    {
      printf ("r2 small in force_particles_function\n");
      printf ("i:%5d  x:%14.5e  y:%14.5e  z:%14.5e\n", i, x[i], y[i], z[i]);
      printf ("j:%5d  x:%14.5e  y:%14.5e  z:%14.5e\n", j, x[j], y[j], z[j]);
      printf ("Press any key ...");
      getch  ();
      printf ("\n");
    }
  if (r2 < r_cutoff_2)
    {
      force_generic = force_generic_function_2 (i, j, r2);
      *force_x = force_generic * dx;
      *force_y = force_generic * dy;
      *force_z = force_generic * dz;
    }
  /* perhaps real particle j is too far away, but virtual particle j is closer */
  else if (dz > 0.0)
  /* implicit assumption: (height of populated region /2) > r_cutoff */
    {
      /* z_virtual = z[j] + 5.0 * tube_height / 7.0; */
      z_virtual = z[j] + tube_height / 3.0;
      dz = z [i] - z_virtual;
      r2 = dx*dx + dy*dy + dz*dz;
      if (r2 < r_cutoff_2)
        {
          force_generic = force_generic_function_2 (i, j, r2);
          *force_x = force_generic * dx;
          *force_y = force_generic * dy;
          *force_z = force_generic * dz;
        }
    }
  /* perhaps real particle j is too far away, but virtual particle j is closer */
  else if (dz < 0.0)
    {
      /* z_virtual = z[j] - 5.0 * tube_height / 7.0; */
      z_virtual = z[j] - tube_height / 3.0;
      dz = z [i] - z_virtual;
      r2 = dx*dx + dy*dy + dz*dz;
      if (r2 < r_cutoff_2)
        {
          force_generic = force_generic_function_2 (i, j, r2);
          *force_x = force_generic * dx;
          *force_y = force_generic * dy;
          *force_z = force_generic * dz;
        }
    }
  return;
}


#if defined (NANOTUBE)
void energy_wall_function (double Xi, double Yi, double Zi, int I, double *En,
double *Vir)
{
  double dx, dy, dz, r2, virial_ij, energy_ij;
  int    j;

  j = 0;
  while ( (z_carbon [j] < (Zi - r_cutoff_wall)) && (j < n_carbon) )
```

```
    j++;

  while ( (z_carbon [j] < (Zi + r_cutoff_wall)) && (j < n_carbon) )
    {
      dx = Xi - x_carbon [j];
      dy = Yi - y_carbon [j];
      dz = Zi - z_carbon [j];
      r2 = dx*dx + dy*dy + dz*dz;

      if (r2 < r_minimum_2/2)
        {
          printf ("r2 small in energy_wall_function\n");
          printf ("Press any key ...");
          getch ();
          printf ("\n");
        }

      /* calculate energy and virial pair i,j */
      if (comp [I] == 1)
        energy_function (&energy_ij, &virial_ij, r2, sigma2_C_1, eps4_C_1,
            eps48_C_1, r_cutoff_wall_2);
      else if (comp [I] == 2)
        energy_function (&energy_ij, &virial_ij, r2, sigma2_C_2, eps4_C_2,
            eps48_C_2, r_cutoff_wall_2);
      #if defined (COMP3)
      else if (comp [I] == 3)
        energy_function (&energy_ij, &virial_ij, r2, sigma2_C_3, eps4_C_3,
            eps48_C_3, r_cutoff_wall_2);
      #endif
      else
        exit (1);
      *En  += energy_ij;
      *Vir += virial_ij;
      j++;
    }
  return;
}


void force_wall_function (int i, int j, double *force_x, double *force_y, double
*force_z)
/*   the sign of the force may be changed by the sign of dx, dy, dz          */
/*   example: if particle i is above carbon atom j, and the force is attractive
(negative), force (i,j) < 0 */
/*   so the function force (i,j) returns the force exerted by j on i   */
{
  double  dx, dy, dz, r2;
  double  force_generic;

  dx = x[i] - x_carbon [j];
  dy = y[i] - y_carbon [j];
  dz = z[i] - z_carbon [j];
  r2 = dx*dx + dy*dy + dz*dz;

  if (r2 < r_minimum_2/2)
    {
      printf ("r2 small in force_wall_function\n");
      printf ("Press any key ...");
      getch ();
      printf ("\n");
    }

  if (comp [i] == 1)
    force_generic = force_generic_function (r2, sigma2_C_1, eps48_C_1);
  else if (comp [i] == 2)
    force_generic = force_generic_function (r2, sigma2_C_2, eps48_C_2);
  #if defined (COMP3)
  else if (comp [i] == 3)
    force_generic = force_generic_function (r2, sigma2_C_3, eps48_C_3);
  #endif
  else
    exit (1);

  *force_x = force_generic * dx;
  *force_y = force_generic * dy;
  *force_z = force_generic * dz;

  return;
}
#endif


/* calculates total energy and virial */
double energy_potential_function (double *Vir)
{
  double xi, yi, zi, energy_i, virial_i;
  double energy_total;
  int    i, j_begin;
  #if defined (TAIL_CORRECT)
    double rho_particles;
  #endif

  energy_total = 0.0;
  *Vir  = 0.0;

  for (i = 0; i < npart-1; i++)
    {
      xi = x [i];
      yi = y [i];
      zi = z [i];

      /* calculate energy particle i with particle j=jb,npart */
      j_begin = i + 1;
      energy_i = 0.0;
      virial_i = 0.0;
      energy_particle_function (xi, yi, zi, i, j_begin, &energy_i, &virial_i);
      energy_total += energy_i;
      *Vir  += virial_i;
    }

  #if defined (NANOTUBE)
  for (i = 0; i < npart; i++)
    {
      xi = x [i];
      yi = y [i];
      zi = z [i];

      /* calculate energy particle i with carbon nanotube */
      energy_i = 0.0;
      virial_i = 0.0;
      energy_wall_function (xi, yi, zi, i, &energy_i, &virial_i);
      energy_total += energy_i;
      *Vir  += virial_i;
    }
  #endif

  /* add tail corrections */
  #if defined (TAIL_CORRECT)
    rho_particles = ((double) npart) /(tube_volume);
    energy_total += npart * correct_u (r_cutoff, rho_particles, sigma3_1_1,
eps4_1_1);
  #endif
  return (energy_total);
}

/***  PARTICLE MANIPULATION  ***/

#if defined (CHECK_PARTICLES_ORDER)
int check_particles_order (int npart, char *name_function)
{
  int i;

  for (i = 0; i < npart - 1; i++)
    {
      if (z [i] > z [i+1])
        {
          printf ("Particles not ordered after %20s !\n", name_function);
          printf ("Press any key\n");
          getch  ();
          return (0);
        }
    }
  return (1);
}
#endif


int check_space_not_ordered (double x_new, double y_new, double z_new, int npart)
{
  int i;

  for (i = 0; i < npart; i++)
    {
      if ( ((x_new - x[i])*(x_new - x[i]) + (y_new - y[i])*(y_new - y[i]) + (z_new
      - z[i])*(z_new - z[i])) < r_minimum_2)
        return (0);
    }

    #if defined (NANOTUBE)
    i = 0;
    while ( (i < n_carbon) && (z_carbon[i] < (z_new - r_minimum)) )
      i++;
    while ( (i < n_carbon) && (z_carbon[i] < (z_new + r_minimum)) )
      {
        if ( ((x_new - x_carbon[i])*(x_new - x_carbon[i])
            + (y_new - y_carbon[i])*(y_new - y_carbon[i])
            + (z_new - z_carbon[i])*(z_new - z_carbon[i])) < r_minimum_2)
          return (0);
        i++;
      }
    #endif

  return (1);
}


#if defined (LATTICE_CYLIND)
void order_particles (double *x, double *y, double *z, int npart)
{
  int     i, j, j_fixed;
  double  z_min;
  double  *x_temp, *y_temp, *z_temp, *x_prev_temp, *y_prev_temp, *z_prev_temp,
      *z_initial_temp;
  double  *x_prev_prev_temp, *y_prev_prev_temp, *z_prev_prev_temp;
  double  *mass_temp;
  int     *comp_temp;

  x_temp           = (double *) malloc (npart * sizeof (double));
  y_temp           = (double *) malloc (npart * sizeof (double));
  z_temp           = (double *) malloc (npart * sizeof (double));
  mass_temp        = (double *) malloc (npart * sizeof (double));
  comp_temp        = (int *)    malloc (npart * sizeof (int));
  x_prev_temp      = (double *) malloc (npart * sizeof (double));
  y_prev_temp      = (double *) malloc (npart * sizeof (double));
  z_prev_temp      = (double *) malloc (npart * sizeof (double));
  x_prev_prev_temp = (double *) malloc (npart * sizeof (double));
  y_prev_prev_temp = (double *) malloc (npart * sizeof (double));
  z_prev_prev_temp = (double *) malloc (npart * sizeof (double));

  z_initial_temp = (double *) malloc (npart * sizeof (double));

  for (i = 0; i < npart; i++)
    {
      z_min = z [0];
      j_fixed = 0;
      for (j = 0; j < npart; j++)
        {
          if (z[j] <= z_min)
            {
              z_min = z [j];
              j_fixed = j;
            }
        }
      x_temp    [i] = x    [j_fixed];
      y_temp    [i] = y    [j_fixed];
      z_temp    [i] = z    [j_fixed];
      comp_temp [i] = comp [j_fixed];
      #if defined (MOLECULAR_DYNAMICS)
        x_prev_temp      [i] = x_prev      [j_fixed];
        y_prev_temp      [i] = y_prev      [j_fixed];
        z_prev_temp      [i] = z_prev      [j_fixed];
        x_prev_prev_temp [i] = x_prev_prev [j_fixed];
        y_prev_prev_temp [i] = y_prev_prev [j_fixed];
        z_prev_prev_temp [i] = z_prev_prev [j_fixed];
        z_initial_temp   [i] = z_initial   [j_fixed];
        mass_temp        [i] = mass        [j_fixed];
      #endif
      z [j_fixed] = 1.0e+24;
    }

  for (i = 0; i < npart; i++)
    {
      x    [i] = x_temp      [i];
      y    [i] = y_temp      [i];
      z    [i] = z_temp      [i];
      comp [i] = comp_temp   [i];
      #if defined (MOLECULAR_DYNAMICS)
        x_prev      [i] = x_prev_temp      [i];
        y_prev      [i] = y_prev_temp      [i];
        z_prev      [i] = z_prev_temp      [i];
        x_prev_prev [i] = x_prev_prev_temp [i];
        y_prev_prev [i] = y_prev_prev_temp [i];
        z_prev_prev [i] = z_prev_prev_temp [i];
        z_initial   [i] = z_initial_temp   [i];
        mass        [i] = mass_temp        [i];
      #endif
    }

  free (x_temp);
  free (y_temp);
  free (z_temp);
  free (x_prev_temp);
  free (y_prev_temp);
  free (z_prev_temp);
  free (x_prev_prev_temp);
  free (y_prev_prev_temp);
  free (z_prev_prev_temp);
  free (z_initial_temp);
  free (mass_temp);
  free (comp_temp);

  #if defined (CHECK_PARTICLES_ORDER)
    check_particles_order (npart, "order_particles");
  #endif
  return;
}

void order_carbon_atoms (double *x_c, double *y_c, double *z_c, int n_carbon)
{
  int     i, j, j_fixed;
  double  z_min;
  double  *x_temp, *y_temp, *z_temp;

  x_temp  = (double *) malloc (n_carbon * sizeof (double));
  y_temp  = (double *) malloc (n_carbon * sizeof (double));
  z_temp  = (double *) malloc (n_carbon * sizeof (double));

  for (i = 0; i < n_carbon; i++)
    {
      z_min = z_c [0];
      j_fixed = 0;
      for (j = 0; j < n_carbon; j++)
        {
          if (z_c [j] < z_min)
            {
```

```
              z_min = z_c [j];
              j_fixed = j;
            }
        }
      x_temp    [i] = x_c  [j_fixed];
      y_temp    [i] = y_c  [j_fixed];
      z_temp    [i] = z_c  [j_fixed];
      z_c [j_fixed] = 1.0e+24;
    }

  for (i = 0; i < n_carbon; i++)
    {
      x_c [i] = x_temp [i];
      y_c [i] = y_temp [i];
      z_c [i] = z_temp [i];
    }

  free (x_temp);
  free (y_temp);
  free (z_temp);
  return;
}


void replace_particle (int old, double x_new, double y_new, double z_new, int
comp_new)
{
  int  i;

  #if defined (CHECK_PARTICLES_ORDER)
    check_particles_order (npart, "replace_particle 0");
  #endif
  if (z [old] < z_new)
    {
      i = old;
      while ( (i < (npart-1)) && (z [i+1] < z_new) )
        {
          z    [i] = z    [i+1];
          y    [i] = y    [i+1];
          x    [i] = x    [i+1];
          comp [i] = comp [i+1];
          i++;
        }
      z    [i] = z_new;
      y    [i] = y_new;
      x    [i] = x_new;
      comp [i] = comp_new;
    }
  else
    {
      i = old;
      while ( (i > 0) && (z [i-1] > z_new) )
        {
          z    [i] = z    [i-1];
          y    [i] = y    [i-1];
          x    [i] = x    [i-1];
          comp [i] = comp [i-1];
          i--;
        }
      z    [i] = z_new;
      y    [i] = y_new;
      x    [i] = x_new;
      comp [i] = comp_new;
    }
  #if defined (CHECK_PARTICLES_ORDER)
    if (!check_particles_order (npart, "replace_particle 1"))
      {
        printf ("Particles not ordered after replace_particle\n");
        printf ("Press any key ...\n");
        getch ();
      }
  #endif
  return;
}


#if defined (GRAND_CANONICAL)
void insert_particle (double x_new, double y_new, double z_new, int comp_new)
{
  int  i;

  i = npart - 1;
  /* z [npart-1] is not yet defined, as npart has just been increased */
  while (z [i-1] > z_new)
    {
      x    [i] = x    [i-1];
      y    [i] = y    [i-1];
      z    [i] = z    [i-1];
      comp [i] = comp [i-1];
      i--;
    }
  x    [i] = x_new;
  y    [i] = y_new;
  z    [i] = z_new;
  comp [i] = comp_new;
  #if defined (CHECK_PARTICLES_ORDER)
    check_particles_order (npart, "insert_particle");
  #endif
  return;
}

#endif
#endif /* LATTICE_CYLIND */

void fill_particle (int i_empty)
{
  int  i;

  for (i = i_empty; i < npart-1; i++)
    {
      x    [i] = x    [i+1];
      y    [i] = y    [i+1];
      z    [i] = z    [i+1];
      comp [i] = comp [i+1];
      #if defined (MOLECULAR_DYNAMICS)
        x_prev      [i] = x_prev      [i+1];
        y_prev      [i] = y_prev      [i+1];
        z_prev      [i] = z_prev      [i+1];
        x_prev_prev [i] = x_prev_prev [i+1];
        y_prev_prev [i] = y_prev_prev [i+1];
        z_prev_prev [i] = z_prev_prev [i+1];
        z_initial   [i] = z_initial   [i+1];
        mass        [i] = mass        [i+1];
      #endif
    }
  #if defined (CHECK_PARTICLES_ORDER)
    check_particles_order (npart, "fill_particle");
  #endif
  return;
}

int check_space (double x_new, double y_new, double z_new, int npart)
{
  int i;

  i = 0;

  while ( (z[i] < (z_new - 2.0 * r_minimum)) && (i < npart) )
    i++;
  while ( (z[i] < (z_new + 2.0 * r_minimum)) && (i < npart) )
    {
      if ( ((x_new - x[i])*(x_new - x[i]) + (y_new - y[i])*(y_new - y[i]) + (z_new
      - z[i])*(z_new - z[i])) < r_minimum_2)
        return (0);
      i++;
    }

    #if defined (NANOTUBE)
    i = 0;
    while ( (i < n_carbon) && (z_carbon[i] < (z_new - 2.0 * r_minimum)) )
      i++;
    while ( (i < n_carbon) && (z_carbon[i] < (z_new + 2.0 * r_minimum)) )
      {
        if ( ((x_new - x_carbon[i])*(x_new - x_carbon[i])
            + (y_new - y_carbon[i])*(y_new - y_carbon[i])
            + (z_new - z_carbon[i])*(z_new - z_carbon[i])) < r_minimum_2)
          return (0);
        i++;
      }
    #endif

  return (1);
}

#if defined (GRAND_CANONICAL) || defined (CANONICAL)

/* Monte-Carlo simulation */
#if defined (LATTICE_CYLIND)
/* attempts to displace a randomly selected particle */
void mcmove (double *En, double *Vir, int *Attempt, int *Nacc, double Dr)
{
  double energy_new, energy_old, x_new, y_new, z_new, virial_old, virial_new;

  #if defined (CHECK_ENERGY)
    double energy_check, virial_check;
  #endif
```

```
int  o, j_begin;

(*Attempt)++;

/* select a particle at random */
o = (int)(npart * random1000 ());
/* calculate old energy configuration of particle o */
j_begin    = 0;
energy_old = 0.0;
virial_old = 0.0;
energy_particle_function (x [o], y [o], z [o], o, j_begin, &energy_old,
    &virial_old);
#if defined (NANOTUBE)
  energy_wall_function   (x [o], y [o], z [o], o, &energy_old, &virial_old);
#endif

/* give particle a random displacement */
do
  {
    x_new = x [o] + (random1000 () - 0.5) * Dr;
    y_new = y [o] + (random1000 () - 0.5) * Dr;
  }
while
  ( ((x_new * x_new) + (y_new * y_new)) > tube_radius_2 );

z_new = z [o] + (random1000 () - 0.5) * Dr;
if (z_new > tube_height)
  z_new -= tube_height;
if (z_new < 0.0)
  z_new += tube_height;

if (check_space (x_new, y_new, z_new, npart))
  {
    /* calculate new energy configuration */
    j_begin = 0;
    energy_new = 0.0;
    virial_new = 0.0;
    energy_particle_function (x_new, y_new, z_new, o, j_begin, &energy_new,
    &virial_new);
    #if defined (NANOTUBE)
    energy_wall_function     (x_new, y_new, z_new, o, &energy_new, &virial_new);
    #endif

    /* acceptance test */
    if ((energy_new < energy_old) || ((random1000 ()) < exp(-beta * (energy_new -
    energy_old))))
      {
        /* accepted */
        (*Nacc)++;
        #if defined (CHECK_ENERGY)
          energy_check = energy_potential_function (&virial_check);
          if ( (fabs (*En - energy_check)) > (fabs ((*En)* 1.0e-03)) )
            getch ();
            /* if ( (fabs (*Vir - virial_check)) > (fabs ((*Vir)* 1.0e-03)) )
              getch () */
        #endif

        *En  += (energy_new - energy_old);
        *Vir += (virial_new - virial_old);
```

```
          replace_particle (o, x_new, y_new, z_new, comp [o]);

          #if defined (CHECK_ENERGY)
            energy_check = energy_potential_function (&virial_check);
            if ((fabs (*En - energy_check)) > (fabs ((*En)* 1.0e-3)))
              getch ();
            /* if ((fabs (*Vir - virial_check)) > (fabs ((*Vir)* 1.0e-03)))
            getch (); */
          #endif
        }
    }
  return;
}
#elif defined (LATTICE_SQUARE)
/* attempts to displace a randomly selected particle */
void mcmove (double *En, double *Vir, int *Attempt, int *Nacc, double Dr)
{
  double energy_new, energy_old, x_new, y_new, z_new, virial_old, virial_new;
  int    o, j_begin;
  #if defined (CHECK_ENERGY)
    double energy_check, virial_check;
  #endif

  (*Attempt)++;

  /* select a particle at random */
  o = (int)(npart * random1000 ());

  /* calculate old energy configuration */
  j_begin = 0;
  energy_particle_function (x [o], y [o], z [o], o, j_begin, &energy_old,
      &virial_old);

  /* give particle a random displacement */
  x_new = x [o] + (random1000 () - 0.5) * Dr;
  y_new = y [o] + (random1000 () - 0.5) * Dr;
  z_new = z [o] + (random1000 () - 0.5) * Dr;

  /* put particle in simulation box */
  /* this is now before the acceptance test */
  if (x_new < 0)
    x_new += box_side;
  if (x_new > box_side)
    x_new -= box_side;
  if (y_new < 0)
    y_new += box_side;
  if (y_new > box_side)
    y_new -= box_side;
  if (z_new < 0)
    z_new += box_side;
  if (z_new > box_side)
    z_new -= box_side;

  /* calculate new energy configuration */
  j_begin = 0;
  energy_particle_function (x_new, y_new, z_new, o, j_begin, &energy_new,
      &virial_new);
  /* acceptance test */
  if ((energy_new < energy_old) || ((random1000 ()) < exp(-beta * (energy_new -
      energy_old))))
    {
      /* accepted */
      (*Nacc)++;
      *En   += energy_new - energy_old;
      *Vir  += virial_new - virial_old;

      x [o] = x_new;
      y [o] = y_new;
      z [o] = z_new;

      #if defined (CHECK_ENERGY)
      energy_check = energy_potential_function (&virial_check);
      if ((fabs (*En - energy_check)) > (fabs ((*En)* 1.0e-3)))
        getch ();
      #endif
    }
  return;
}
#endif
#endif /* GRAND_CANONICAL || CANONICAL */

#if defined (WIDOM)
/*  calculate excess chemical potential           */
/*  widom never checked with cylindrical lattice  */
double widom (int component)
{
  double  energy, virial, x_new, y_new, z_new;
  #if defined (LATTICE_CYLIND)
    double  r_new, theta_new;
  #endif
  int  old, j_begin, check_space_flag;

  check_space_flag = 0;
  while (!check_space_flag)
    {
      /* add a particle at a random position */
      #if defined (LATTICE_CYLIND)
        r_new     = sqrt (random1000 ()) * tube_radius;
        theta_new = random1000 () * 2.0 * M_PI;

        x_new = r_new * cos (theta_new);
        y_new = r_new * sin (theta_new);
        z_new = random1000 () * tube_height;
      #elif defined (LATTICE_SQUARE)
        x_new = random1000 () * box_side;
        y_new = random1000 () * box_side;
        z_new = random1000 () * box_side;
      #endif

      if (check_space (x_new, y_new, z_new, npart))
        {
          check_space_flag = 1;
          old = npart;  /* index of new particle */
          comp [npart] = component;

          /*  determine energy of this particle */
          j_begin = 0;
          energy  = 0.0;
          virial  = 0.0;
          energy_particle_function (x_new, y_new, z_new, old, j_begin, &energy,
      &virial);
        }
    }
  return (energy);
}
#endif  /* WIDOM */


#if defined (GRAND_CANONICAL)
/* attempts to exchange a particle with the reservoir */
void mcexch (double *En, double *Vir, int *Attempt, int *Nacc)
{
  double  r_new, theta_new, energy_new, energy_old, x_new, y_new, z_new,
          virial_old, virial_new, arg;
  #if defined (TAIL_CORRECT)
  double  rho_particles_old, rho_particles_new;
  #endif
  /* int     comp_new; */
  #if defined (CHECK_ENERGY)
    double energy_check, virial_check;
  #endif
  int     old, j_begin;
  double  x_rand1, x_rand2;
  double  zz_local;
  int     npart_local;

  (*Attempt)++;
  #if defined (TAIL_CORRECT)
    rho_particles_old = ((double) npart) / tube_volume;
  #endif

  /* select to add or delete a particle */
  x_rand1 = random1000 ();
  if (x_rand1 < 0.5)
    {
      /* add a particle at a random position */
      r_new     = sqrt (random1000 ()) * tube_radius;
      theta_new = random1000 () * 2.0 * M_PI;

      x_new = r_new * cos (theta_new);
      y_new = r_new * sin (theta_new);
      z_new = random1000 () * tube_height;

      if (check_space (x_new, y_new, z_new, npart))
        {
          old = npart;  /* index of new particle */
          x_rand2 = random1000 ();
          if (x_rand2 < RATIO_COMP1)
            {
              comp [npart] = 1;
              zz_local     = zz_1;
            }
          #if defined (COMP3)
          else if (x_rand2 > (RATIO_COMP1 + RATIO_COMP2))
            {
```

```
    comp [npart] = 3;
    zz_local     = zz_3;
  }
#endif
else
  {
    comp [npart] = 2;
    zz_local     = zz_2;
  }
npart_local = npart_comp [comp [npart]];
/*  determine energy of this particle   */
j_begin    = 0;
energy_new = 0.0;
virial_new = 0.0;
energy_particle_function (x_new, y_new, z_new, old, j_begin, &energy_new,
  &virial_new);
#if defined (NANOTUBE)
  energy_wall_function   (x_new, y_new, z_new, old, &energy_new,
  &virial_new);
#endif

/* tail correction */
#if defined (TAIL_CORRECT)
  rho_particles_new = ((double)(npart+1)) / tube_volume;
  energy_new  = energy_new
    + ((npart+1) * correct_u (r_cutoff, rho_particles_new, sigma3_1_1,
  eps4_1_1)
    -   npart    * correct_u (r_cutoff, rho_particles_old, sigma3_1_1,
  eps4_1_1));
#endif

/* acceptance test */
if ((-beta * energy_new) >= 10.0)
  arg = 1.01;
else
  {
    if ((-beta*energy_new) < -500.0)
      arg = 0.0;
    else
      arg = zz_local * tube_volume * exp(-beta * energy_new) /
        (npart_local+1);
  }
if ((random1000 ()) < arg)
  {
    /* accepted */
    (*Nacc)++;
    #if defined (CHECK_ENERGY)
      energy_check = energy_potential_function (&virial_check);
      if ((fabs (*En - energy_check)) > (fabs ((*En)* 1.0e-03)))
        getch ();
      /* if ((fabs (*Vir - virial_check)) > (fabs ((*Vir)* 1.0e-03)))
        getch (); */
    #endif
    *En  += energy_new;
    *Vir += virial_new;

    npart_comp [comp [npart]]++;
    npart++;
    /* printf ("npart: %5d\n", npart); */
```

```
            if (npart > NPART_MAX)
              {
                printf ("Too many particles !");
                printf ("npart: %5d", npart);
                exit (1);
              }

            x [npart-1] = x_new;
            y [npart-1] = y_new;
            z [npart-1] = z_new;
            /* insert_particle (x_new, y_new, z_new, comp_new); */
            order_particles (x, y, z, npart);

            #if defined (CHECK_ENERGY)
              energy_check = energy_potential_function (&virial_check);
              if ((fabs (*En - energy_check)) > (fabs ((*En)* 1.0e-03)))
                getch ();
              /* if ( (fabs (*Vir - virial_check)) > (fabs ((*Vir)* 1.0e-03)) )
                getch (); */
            #endif
          }
      }
  }
else
  {
    if (npart == 0)
      return;

    /* delete a randomly selected particle */
    old = (int)(random1000 () * npart);
    if (comp [old] == 1)
      zz_local = zz_1;
    else if (comp [old] == 2)
      zz_local = zz_2;
    #if defined (COMP3)
    else if (comp [old] == 3)
      zz_local = zz_3;
    #endif
    else
      exit (1);

    npart_local = npart_comp [comp [old]];

    j_begin = 0;
    energy_old = 0.0;
    virial_old = 0.0;
    energy_particle_function (x[old], y[old], z[old], old, j_begin, &energy_old,
    &virial_old);
    #if defined (NANOTUBE)
      energy_wall_function   (x[old], y[old], z[old], old, &energy_old,
          &virial_old);
    #endif

    /* particle is removed, so new energy */
    energy_new = -energy_old;
    virial_new = -virial_old;

    /* tail correction */
    #if defined (TAIL_CORRECT)
```

```
        rho_particles_new = ((double)(npart-1)) / tube_volume;
        energy_new = energy_new
          + ((npart-1) * correct_u (r_cutoff, rho_particles_new, sigma3_1_1,
            eps4_1_1)
          -   npart    * correct_u (r_cutoff, rho_particles_old, sigma3_1_1,
            eps4_1_1));
      #endif

      /* acceptance test */
      if ((-beta * energy_new) >= 10.0)
        arg = 1.01;
      else
        arg = npart_local * exp (-beta * energy_new) / (zz_local * tube_volume);
      if ((random1000 ()) < arg)
        {
          /* accepted */
          (*Nacc)++;
          npart_comp [comp [old]]--;

          #if defined (CHECK_ENERGY)
            energy_check = energy_potential_function (&virial_check);
            if ((fabs (*En - energy_check)) > (fabs ((*En)* 1.0e-03)))
              getch ();
            /* if ( (fabs (*Vir - virial_check)) > (fabs ((*Vir)* 1.0e-03)) )
              getch (); */
          #endif
          *En  += energy_new;
          *Vir += virial_new;
          fill_particle (old);
          npart--;

          #if defined (CHECK_ENERGY)
            energy_check = energy_potential_function (&virial_check);
            if ((fabs (*En - energy_check)) > (fabs ((*En)* 1.0e-03)))
              getch ();
            /* if ( (fabs (*Vir - virial_check)) > (fabs ((*Vir)* 1.0e-03)) )
              getch (); */
          #endif
        }
    }
  return;
}
#endif  /* PRESSURE_ACTUAL */


#if defined (GRAND_CANONICAL) || defined (CANONICAL)
#if defined (LATTICE_CYLIND)
/* place 'npart' particles on a lattice */
#if defined (COMP3)
  int lattice (int *npart_comp, int comp_1st, int comp_2nd, int comp_3rd)
#else
  int lattice (int *npart_comp, int comp_1st, int comp_2nd)
#endif
{
  double x_new, y_new, z_new;
  double r_new, theta_new;
  int    npart_temp [3];
  int    comp_new;
  int    i, j, k;
```

```
npart_temp [1] = (int) (RATIO_COMP1 * npart);
npart_temp [2] = (int) (RATIO_COMP2 * npart);

npart_comp [1] = 0;
npart_comp [2] = 0;
#if defined (COMP3)
  npart_comp [3] = 0;
#endif


i = 0;
while (i < npart)
  {
    r_new     = sqrt (random1000 ()) * tube_radius;
    theta_new = random1000 () * 2.0 * M_PI;
    z_new     = random1000 () * tube_height;

    x_new = r_new * cos (theta_new);
    y_new = r_new * sin (theta_new);

    if (i < npart_temp [comp_1st])
      comp_new = comp_1st;
    #if defined (COMP3)
    else if (i < (npart_temp [comp_1st] + npart_temp [comp_2nd]))
      comp_new = comp_2nd;
    else
      comp_new = comp_3rd;
    #else
    else
      comp_new = comp_2nd;
    #endif

    if ((i == 0) || (check_space_not_ordered (x_new, y_new, z_new, i)))
      {
        x    [i] = x_new;
        y    [i] = y_new;
        z    [i] = z_new;
        comp [i] = comp_new;
        npart_comp [comp[i]]++;
        i++;

        #if defined (NANOTUBE)
        for (j = 0; j < n_carbon; j++)
          {
            for (k = 0; k < i; k++)
              {
                if ( ((x_carbon[j]-x[k])*(x_carbon[j]-x[k])+(y_carbon[j]-
                      y[k])*(y_carbon[j]-y[k])+(z_carbon[j]-z[k])*(z_carbon[j]-
                      z[k])) < r_minimum_2)
                  {
                    printf ("r2 small in lattice - carbon\n");
                    getch ();
                    exit (1);
                  }
              }
          }
        #endif
      }
    }
  return (i);
}
#elif defined (LATTICE_SQUARE)
/* to check the code, the components have to be placed on the lattice
   always in the same order */
#if defined (COMP3)
  int lattice (int *npart_comp, int comp_1st, int comp_2nd, int comp_3rd)
#else
  int lattice (int *npart_comp, int comp_1st, int comp_2nd)
#endif
{
  int     i, comp_new;
  double  x_new, y_new, z_new;
  /*
  int     j, k, itel, n;
  double  dx, dy, dz, del;
  double  rnd;
  */

  npart_comp [1] = (int) (RATIO_COMP1 * npart);
  npart_comp [2] = (int) (RATIO_COMP2 * npart);
  #if defined (COMP3)
    npart_comp [3] = (int) ((1.0 - RATIO_COMP1 - RATIO_COMP2) * npart);
  #endif

  i = 0;
  while (i < npart)
    {
      x_new = random1000 () * box_side;
      y_new = random1000 () * box_side;
      z_new = random1000 () * box_side;

      if (i < npart_comp [comp_1st])
        comp_new = comp_1st;
      #if defined (COMP3)
      else if (i < (npart_comp [comp_1st] + npart_comp [comp_2nd]))
        comp_new = comp_2nd;
      else
        comp_new = comp_3rd;
      #else
      else
        comp_new = comp_2nd;
      #endif

      if ((i == 0) || (check_space_not_ordered (x_new, y_new, z_new, i)))
        {
          x    [i] = x_new;
          y    [i] = y_new;
          z    [i] = z_new;
          comp [i] = comp_new;
          i++;
        }
    }

  return (i);
}
#endif
#endif /* GRAND_CANONICAL || CANONICAL */
```

```
#if defined (GRAND_CANONICAL) || defined (CANONICAL)
/* adjusts maximum displacement such that 50% of the moves will be accepted */
void adjust (int Attempt, int Nacc, double *Dr, int cycle)
{
  static int attempt_static, nacc_static;
  double     dro, frac;
  FILE       *out_current;

  if ((Attempt == 0) || (attempt_static >= Attempt))
    {
      nacc_static    = Nacc;
      attempt_static = Attempt;
    }
  else
    {
      frac = ((double)(Nacc - nacc_static))/((double)(Attempt - attempt_static));
      dro = *Dr;
      *Dr = *Dr * fabs (frac/0.5);

      /* limit the change */
      if (*Dr/dro > 1.5)
        *Dr = dro * 1.5;
      if (*Dr/dro < 0.5)
        *Dr = dro * 0.5;
      #if defined (LATTICE_CYLIND)
        if (*Dr > tube_radius / 2.0)
          *Dr = tube_radius / 2.0;
      #elif defined (LATTICE_SQUARE)
        if (*Dr > box_half / 2.0)
          *Dr = box_half / 2.0;
      #endif

      printf ("Max. displ. set to: %10.5e\n", *Dr);
      printf ("(old: %10.5e ')'", dro);
      printf ("Frac. acc.: %4.2f\n", frac);
      printf ("attempts: %7d\n", Attempt - attempt_static);
      printf ("succes: %7d\n", Nacc - nacc_static);

      out_current = fopen ("current.txt", "a+");
      fprintf (out_current, "Cycle: %5d\n");
      fprintf (out_current, "Maximum displacement set to: %10.5e\n", *Dr);
      fprintf (out_current, "(old: %10.5e ')'", dro);
      fprintf (out_current, "Fraction accepted: %4.2f\n", frac);
      fprintf (out_current, "attempts: %7d\n", Attempt - attempt_static);
      fprintf (out_current, "succes: %7d\n", Nacc - nacc_static);
      fprintf (out_current, "\n");
      fclose (out_current);

      /* store nacc and attemp for next use */
      nacc_static    = Nacc;
      attempt_static = Attempt;
    }
  return;
}
#endif
```

```
/* write configuration to disk - ascii file */
void store_asc (FILE *out, double Dr, int i_cycl, int component, int npart_comp)
{
  int  i;

  fprintf (out, "%5d\n",    npart_comp);
  fprintf (out, "%10.5f\n", Dr);
  fprintf (out, "%10d\n",   i_cycl);
  fprintf (out, "%i\n",     component);
  for (i = 0; i < npart; i++)
    {
      if (comp [i] == component)
        fprintf (out, "%10.5e %10.5e %10.5e\n", x[i], y[i], z[i]);
    }
  return;
}


/* write configuration to disk - binary file */
void store_bin (int out, double Dr, int i_cycl, int component)
{
  int  i;

  /* write (out, &box,   sizeof (double)); */
  write (out, &npart,     sizeof (int));
  write (out, &Dr,        sizeof (double));
  write (out, &i_cycl,    sizeof (int));
  write (out, &component, sizeof (int));
  for (i = 0; i < npart; i++)
    {
      if (*(comp+i) == component)
        {
          write (out, x+i, sizeof (double));
          write (out, y+i, sizeof (double));
          write (out, z+i, sizeof (double));
        }
    }
  return;
}


/****************************************************************************/
/*                                                                          */
/*   INPUT DATA & MODEL PARAMETERS                                          */
/*                                                                          */
/****************************************************************************/

/* read configuration from disk - ascii file */
int read_asc (FILE *inp, int component, int npart_begin)
{
  int   i;
  int   npart_comp_local;
  char  npart_comp_str [15], Dr_str [15], i_cycl_str [15], component_str [15];
  char  x_str [15], y_str [15], z_str [15];

  fscanf (inp, "%5s\n", npart_comp_str);
  npart_comp_local = atoi (npart_comp_str);

  fscanf (inp, "%10s\n", Dr_str);
  fscanf (inp, "%10s\n", i_cycl_str);
  fscanf (inp, "%10s\n", component_str);

  for (i = npart_begin; i < npart_comp_local+npart_begin; i++)
    {
      fscanf (inp, "%12s %13s %12s\n", x_str, y_str, z_str);

      x    [i] = atof (x_str);
      y    [i] = atof (y_str);
      z    [i] = atof (z_str);
      comp [i] = component;
      #if defined (MOLECULAR_DYNAMICS)
      switch (component)
        {
          case 1:
            mass [i] = mass_particle_1;
            break;
          case 2:
            mass [i] = mass_particle_2;
            break;
          #if defined (COMP3)
          case 3:
            mass [i] = mass_particle_3;
            break;
          #endif
        }
      #endif
    }
  return (npart_comp_local);
}


#if defined (NANOTUBE)
/* read coordinates of C atoms from binary file */
int get_carbon_coordinates_bin (void)
{
  int  n_carbon_atoms;
  long read_bytes;
  int  inp_bin, i;

  /* read coordinates of C atoms */
  /* output in binary file       */
  inp_bin = open ("tubexyz.bin", O_RDONLY, S_IREAD);
  read_bytes = read (inp_bin, &n_carbon_atoms, 4);
  x_carbon = (double *) malloc (n_carbon_atoms * sizeof (double));
  y_carbon = (double *) malloc (n_carbon_atoms * sizeof (double));
  z_carbon = (double *) malloc (n_carbon_atoms * sizeof (double));

  for (i = 0; i < n_carbon_atoms; i++)
    {
      read_bytes = read (inp_bin, x_carbon+i, sizeof (double));
        *(x_carbon+i) *= 1.0e-10;
      read_bytes = read (inp_bin, y_carbon+i, sizeof (double));
        *(y_carbon+i) *= 1.0e-10;
      read_bytes = read (inp_bin, z_carbon+i, sizeof (double));
        *(z_carbon+i) *= 1.0e-10;
    }
  close (inp_bin);
  return (n_carbon_atoms);
```

```
}

/* read coordinates of C atoms from text file */
int get_c_coordinates_txt (void)
{
  int   n_carbon_atoms;
  int   i;
  FILE  *inp_txt;
  char  n_carbon_atoms_str [10], x_str [15], y_str [15], z_str [15];

  /* read coordinates of C atoms */
  /* output in text file       */
  inp_txt = fopen ("tubexyz.txt", "r+");
  fscanf (inp_txt, "%5s\n", n_carbon_atoms_str);
  n_carbon_atoms = atoi (n_carbon_atoms_str);
  x_carbon = (double *) malloc (n_carbon_atoms * sizeof (double));
  y_carbon = (double *) malloc (n_carbon_atoms * sizeof (double));
  z_carbon = (double *) malloc (n_carbon_atoms * sizeof (double));

  for (i = 0; i < n_carbon_atoms; i++)
    {
      fscanf (inp_txt, "%10s", x_str);
        *(x_carbon+i) = atof (x_str);
        *(x_carbon+i) *= 1.0e-10;
      fscanf (inp_txt, "%11s", y_str);
        *(y_carbon+i) = atof (y_str);
        *(y_carbon+i) *= 1.0e-10;
      fscanf (inp_txt, "%11s\n", z_str);
        *(z_carbon+i) = atof (z_str);
        *(z_carbon+i) *= 1.0e-10;
      printf ("%5d%12.5e%12.5e%12.5e\n", i, *(x_carbon+i), *(y_carbon+i),
*(z_carbon+i));
    }
  printf ("\n");
  fclose (inp_txt);
  return (n_carbon_atoms);
}
#endif


/***  read configuration from files  ***/
#if defined (COMP3)
  int read_config_files (int *npart_comp,
    int comp1, char *file_1,
    int comp2, char *file_2,
    int comp3, char *file_3)
#else
  int read_config_files (int *npart_comp,
    int comp1, char *file_1,
    int comp2, char *file_2)
#endif
{
  FILE    *inp_asc;

  inp_asc = fopen (file_1, "r");
  npart_comp [1] = read_asc (inp_asc, comp1, 0);
  fclose (inp_asc);
  npart = npart_comp [1];
  inp_asc = fopen (file_2, "r");
  npart_comp [2] = read_asc (inp_asc, comp2, npart);
  /* npart is still equal to npart_comp[1] */
  fclose (inp_asc);
  npart += npart_comp [2];

  #if defined (COMP3)
    inp_asc = fopen (file_3, "r");
    npart_comp [3] = read_asc (inp_asc, comp3, npart);
    /* npart is still equal to npart_comp[1]+npart_comp[2] */
    fclose (inp_asc);
    npart += npart_comp [3];
  #endif

  #if defined (MOLECULAR_DYNAMICS)
    x_prev      = (double *) malloc (npart * sizeof (double));
    y_prev      = (double *) malloc (npart * sizeof (double));
    z_prev      = (double *) malloc (npart * sizeof (double));
    x_prev_prev = (double *) malloc (npart * sizeof (double));
    y_prev_prev = (double *) malloc (npart * sizeof (double));
    z_prev_prev = (double *) malloc (npart * sizeof (double));
    z_initial   = (double *) malloc (npart * sizeof (double));
  #endif

  /* sort particles */
  order_particles (x,y,z,npart);

  #if defined (MOLECULAR_DYNAMICS)
    free (x_prev);
    free (y_prev);
    free (z_prev);
    free (x_prev_prev);
    free (y_prev_prev);
    free (z_prev_prev);
    free (z_initial);
  #endif

  return (npart);
}

/*********************************   set_data   ********************************/
#if defined (GRAND_CANONICAL)
  #if defined (COMP3)
    void set_data (int *Equil, int *Prod, int *Nsamp, int *Ndispl, double *Dr,
      int *Nexch, char component_1, char component_2, char component_3)
  #else
    void set_data (int *Equil, int *Prod, int *Nsamp, int *Ndispl, double *Dr,
      int *Nexch, char component_1, char component_2)
  #endif
#elif defined (CANONICAL)
  #if defined (COMP3)
    void set_data (int *Equil, int *Prod, int *Nsamp, int *Ndispl, double *Dr,
      char component_1, char component_2, char component_3)
  #else
    void set_data (int *Equil, int *Prod, int *Nsamp, int *Ndispl, double *Dr,
      char component_1, char component_2)
  #endif
#elif defined (MOLECULAR_DYNAMICS)
  #if defined (COMP3)
```

```
    void set_data (char component_1, char component_2, char component_3)
  #else
    void set_data (char component_1, char component_2)
  #endif
#endif
{
  #if defined (GRAND_CANONICAL) || defined (CANONICAL)
    int nplaced;
  #endif

  #if defined (SHIFT_POTENT)
    double vir;
  #endif
  double sigma_1_1, sigma_2_2, sigma_1_2;
  double epsilon_over_kb_1_1, eps_1_1;
  double epsilon_over_kb_2_2, eps_2_2;
  double epsilon_over_kb_1_2, eps_1_2;
  #if defined (COMP3)
    double sigma_3_3, sigma_1_3, sigma_2_3;
    double epsilon_over_kb_3_3, eps_3_3;
    double epsilon_over_kb_1_3, eps_1_3;
    double epsilon_over_kb_2_3, eps_2_3;
  #endif

  #if defined (NANOTUBE)
    double sigma_C_C, sigma_C_1, sigma_C_2;
    double eps_C_1, eps_C_2;
    double epsilon_over_kb_C_C;
    double epsilon_over_kb_C_1;
    double epsilon_over_kb_C_2;
    #if defined (COMP3)
      double sigma_C_3;
      double eps_C_3;
      double epsilon_over_kb_C_3;
    #endif
  #endif

  #if defined (GRAND_CANONICAL)
    double  mu_1_0, mu_2_0;
    #if defined (COMP3)
      double mu_3_0;
    #endif
  #elif defined (CANONICAL)
    double pressure_actual;
  #endif

  /* input parameters                                                      */
  /*   equil   = number of equilibration cycles                            */
  /*   prod    = number of production cycles                               */
  /*   nsample = sample frequency; sample after nsample cycles             */
  /*   dr      = maximum displacement                                      */
  /*   ndispl  = number of attempts to displace a particle per cycle       */
  /*   nexch   = number of attempts to exchange particles per cycle        */
  /*   npart   = total number of particles                                 */
  /*   temp    = temperature                                               */
  /*   pressure= ideal gas pressure reservoir (transfered to chemical potential) */
```

```
/***  simulation parameters  ***/
#if defined (GRAND_CANONICAL) || defined (CANONICAL)
  FILE  *out_current;

  *Equil  = EQUIL_CYCLES;  /* Equilibration cycles */
  *Prod   = PROD_CYCLES;   /* Production cycles    */
  *Nsamp  = 2;
  *Ndispl = DISPLACEM;
  #if defined (GRAND_CANONICAL)
    *Nexch  = EXCHANGES;
  #endif
#endif

/***  physical parameters  ***/
temp = TEMPERATURE_ACTUAL;        /* K  */

#if defined (CANONICAL)
  pressure_actual = PRESSURE_ACTUAL;           /* Pa */
#endif
#if defined (GRAND_CANONICAL) || defined (WIDOM)
  pressure = PRESSURE_ACTUAL;           /* Pa */
  beta     = 1.0 / (BOLTZMANN * temp);
  mu_ideal_gas = (R_GAS * temp / AVOGADRO) * log (pressure / 1.0e+05);
#endif

#if defined (MOLECULAR_DYNAMICS)
  dt     = DT;
  t_max  = TIME_EQUIL+TIME_AVER;
  dt2    = dt * dt;
  two_dt = 2.0 * dt;
#endif

/***  geometrical parameters  ***/
#if defined (LATTICE_CYLIND)
  tube_height   = TUBE_HEIGHT;
  tube_radius   = TUBE_DIAMETER / 2.0;
  tube_radius_2 = tube_radius * tube_radius;
  tube_volume   = M_PI * tube_radius * tube_radius * tube_height;
#elif defined (LATTICE_SQUARE)
  box_side    = BOX_SQUARE;
  box_half    = box_side / 2.0;
  tube_volume = box_side * box_side * box_side;
#endif


/***  number of particles  ***/
/*    GC: first maximum, then initial */
/*    C, MD : fixed                   */
#if defined (CANONICAL)
  npart = (int) ( (pressure * tube_volume  / (R_GAS * temp)) * AVOGADRO );
#elif defined (GRAND_CANONICAL) || defined (MOLECULAR_DYNAMICS)
  npart = NPART_MAX;
#endif

x    = (double *) malloc ((npart+1) * sizeof (double));
/* npart+1, so that widom may always be used */
y    = (double *) malloc ((npart+1) * sizeof (double));
z    = (double *) malloc ((npart+1) * sizeof (double));
comp = (int *)    malloc ((npart+1) * sizeof (double));
```

```
#if defined (MOLECULAR_DYNAMICS)
  mass = (double *) malloc ((npart+1) * sizeof (double));

  /*
  x_prev = (double *) malloc (npart * sizeof (double));
  y_prev = (double *) malloc (npart * sizeof (double));
  z_prev = (double *) malloc (npart * sizeof (double));

  v_x = (double *) malloc (npart * sizeof (double));
  v_y = (double *) malloc (npart * sizeof (double));
  v_z = (double *) malloc (npart * sizeof (double));
  */
#endif

#if defined (GRAND_CANONICAL)
  npart = NPART_INITIAL;  /* after allocation of sufficient memory */
#endif

/***  generate configuration  ***/
#if defined (GRAND_CANONICAL) || defined (CANONICAL)
  npart_comp [1] = 0;
  npart_comp [2] = 0;
  #if defined (COMP3)
    npart_comp [3] = 0;
    nplaced = lattice (npart_comp, 1, 2, 3);
  #else
    nplaced = lattice (npart_comp, 1, 2);
  #endif
  if (nplaced < npart)
    exit (1);

  #if defined (LATTICE_CYLIND)
    order_particles (x,y,z,npart);
  #endif

  /***  particle densities  ***/
  rho_particles_1 = npart_comp [1] / tube_volume;
  rho_particles_2 = npart_comp [2] / tube_volume;
  #if defined (COMP3)
    rho_particles_3 = npart_comp [3] / tube_volume;
  #endif
#endif

/***  sigma, epsilon, molecular mass  ***/
#if defined (NANOTUBE)
  /* C - C */
  sigma_C_C = SIGMA_CC;
  epsilon_over_kb_C_C = EPSILON_OVER_KB_CC;
#endif

switch (component_1)
  {
    case 'H':
      sigma_1_1           = SIGMA_H2;
      epsilon_over_kb_1_1 = EPSILON_OVER_KB_H2;
      mass_molecular_1    = MOLECULAR_MASS_H2;
      break;
    case 'N':
      sigma_1_1           = SIGMA_N2;
      epsilon_over_kb_1_1 = EPSILON_OVER_KB_N2;
      mass_molecular_1    = MOLECULAR_MASS_N2;
      break;
    case 'O':
      sigma_1_1           = SIGMA_O2;
      epsilon_over_kb_1_1 = EPSILON_OVER_KB_O2;
      mass_molecular_1    = MOLECULAR_MASS_O2;
      break;
    case 'C':
      sigma_1_1           = SIGMA_CO2;
      epsilon_over_kb_1_1 = EPSILON_OVER_KB_CO2;
      mass_molecular_1    = MOLECULAR_MASS_CO2;
      break;
    case 'M':
      sigma_1_1           = SIGMA_CH4;
      epsilon_over_kb_1_1 = EPSILON_OVER_KB_CH4;
      mass_molecular_1    = MOLECULAR_MASS_CH4;
      break;
  }

sigma2_1_1 = sigma_1_1 * sigma_1_1;
sigma3_1_1 = sigma_1_1 * sigma_1_1 * sigma_1_1;

eps_1_1   = epsilon_over_kb_1_1 * BOLTZMANN;
eps4_1_1  = 4.0  * eps_1_1;
eps48_1_1 = 48.0 * eps_1_1;

#if defined (GRAND_CANONICAL)
  lambda_1 = PLANCK / sqrt (2.0 * M_PI * (mass_molecular_1 / AVOGADRO) *
    BOLTZMANN * temp);
  mu_1_0 = 0.0;  /* 455.58e+03 / AVOGADRO ;  /* [J/particle] */
  mu_1   = mu_1_0 + R_GAS * temp * log (pressure * RATIO_COMP1 / 1.0e+05) /
    AVOGADRO; /* [J/particle] */
  zz_1   = exp (mu_1/(BOLTZMANN * temp))/(lambda_1 * lambda_1 * lambda_1);
  /* temporary */
  zz_1   = beta * pressure * RATIO_COMP1;
#endif

#if defined (GRAND_CANONICAL) || defined (CANONICAL)
  *Dr = DR_INIT; /* not really important at the beginning */
#endif

/* cutoff distance calculated now, because it may depend on sigma */
r_minimum   = R_MINIMUM;
r_minimum_2 = r_minimum * r_minimum;
#if defined (CUTOFF_DISTANCE)
  r_cutoff      = CUTOFF_DISTANCE;
  r_cutoff_wall = CUTOFF_DISTANCE;
#elif !defined (CUTOFF_DISTANCE)
  r_cutoff      = 5.0 * sigma_1_1;
  r_cutoff_wall = 5.0 * sigma_1_1;  /* 1.5 * tube_radius; */
#endif

/* for both defined(CUTOFF_DISTANCE) and !defined(CUTOFF_DISTANCE) */
r_cutoff_2      = r_cutoff * r_cutoff;
r_cutoff_wall_2 = r_cutoff_wall * r_cutoff_wall;

/* C - 1 */
#if defined (NANOTUBE)
```

```
  sigma_C_1  = (sigma_C_C + sigma_1_1) / 2;
  sigma2_C_1 = sigma_C_1 * sigma_C_1;
  sigma3_C_1 = sigma_C_1 * sigma_C_1 * sigma_C_1;

  epsilon_over_kb_C_1 = sqrt (epsilon_over_kb_C_C * epsilon_over_kb_1_1);
  eps_C_1   = epsilon_over_kb_C_1 * BOLTZMANN;
  eps4_C_1  = 4.0  * eps_C_1;
  eps48_C_1 = 48.0 * eps_C_1;
#endif

switch (component_2)
  {
    case 'H':
      sigma_2_2           = SIGMA_H2;
      epsilon_over_kb_2_2 = EPSILON_OVER_KB_H2;
      mass_molecular_2    = MOLECULAR_MASS_H2;
      break;
    case 'N':
      sigma_2_2           = SIGMA_N2;
      epsilon_over_kb_2_2 = EPSILON_OVER_KB_N2;
      mass_molecular_2    = MOLECULAR_MASS_N2;
      break;
    case 'O':
      sigma_2_2           = SIGMA_O2;
      epsilon_over_kb_2_2 = EPSILON_OVER_KB_O2;
      mass_molecular_2    = MOLECULAR_MASS_O2;
      break;
    case 'C':
      sigma_2_2           = SIGMA_CO2;
      epsilon_over_kb_2_2 = EPSILON_OVER_KB_CO2;
      mass_molecular_2    = MOLECULAR_MASS_CO2;
      break;
    case 'M':
      sigma_2_2           = SIGMA_CH4;
      epsilon_over_kb_2_2 = EPSILON_OVER_KB_CH4;
      mass_molecular_2    = MOLECULAR_MASS_CH4;
      break;
  }
sigma2_2_2 = sigma_2_2 * sigma_2_2;
sigma3_2_2 = sigma_2_2 * sigma_2_2 * sigma_2_2;

eps_2_2   = epsilon_over_kb_2_2 * BOLTZMANN;
eps4_2_2  = 4.0  * eps_2_2;
eps48_2_2 = 48.0 * eps_2_2;

#if defined (GRAND_CANONICAL)
  lambda_2 = PLANCK / sqrt (2.0 * M_PI * (mass_molecular_2 / AVOGADRO) *
    BOLTZMANN * temp);
  mu_2_0   = 0.0;  /* 455.58e+03 / AVOGADRO ;  /* [J/particle] */
  /* mu_2 = mu_2_0 + R_GAS * temp * log (pressure * RATIO_COMP2 / 1.0e+05) /
    AVOGADRO; */ /* [J/particle] */
  /* zz_2 = exp (mu_2/(BOLTZMANN * temp))/(lambda_2 * lambda_2 * lambda_2); */
  /* temporary */
  zz_2     = beta * pressure * RATIO_COMP2;
#endif

#if defined (NANOTUBE)
  /* C - 2 */
  sigma_C_2 = (sigma_C_C + sigma_2_2) / 2;
  sigma2_C_2 = sigma_C_2 * sigma_C_2;
  sigma3_C_2 = sigma_C_2 * sigma_C_2 * sigma_C_2;

  epsilon_over_kb_C_2 = sqrt (epsilon_over_kb_C_C * epsilon_over_kb_2_2);
  eps_C_2 = epsilon_over_kb_C_2 * BOLTZMANN;
  eps4_C_2  = 4.0  * eps_C_2;
  eps48_C_2 = 48.0 * eps_C_2;
#endif


#if defined (COMP3)
switch (component_3)
  {
    case 'H':
      sigma_3_3           = SIGMA_H2;
      epsilon_over_kb_3_3 = EPSILON_OVER_KB_H2;
      mass_molecular_3    = MOLECULAR_MASS_H2;
      break;
    case 'N':
      sigma_3_3           = SIGMA_N2;
      epsilon_over_kb_3_3 = EPSILON_OVER_KB_N2;
      mass_molecular_3    = MOLECULAR_MASS_N2;
      break;
    case 'O':
      sigma_3_3           = SIGMA_O2;
      epsilon_over_kb_3_3 = EPSILON_OVER_KB_O2;
      mass_molecular_3    = MOLECULAR_MASS_O2;
      break;
    case 'C':
      sigma_3_3           = SIGMA_CO2;
      epsilon_over_kb_3_3 = EPSILON_OVER_KB_CO2;
      mass_molecular_3    = MOLECULAR_MASS_CO2;
      break;
    case 'M':
      sigma_3_3           = SIGMA_CH4;
      epsilon_over_kb_3_3 = EPSILON_OVER_KB_CH4;
      mass_molecular_3    = MOLECULAR_MASS_CH4;
      break;
  }
sigma2_3_3 = sigma_3_3 * sigma_3_3;
sigma3_3_3 = sigma_3_3 * sigma_3_3 * sigma_3_3;

eps_3_3   = epsilon_over_kb_3_3 * BOLTZMANN;
eps4_3_3  = 4.0  * eps_3_3;
eps48_3_3 = 48.0 * eps_3_3;

#if defined (GRAND_CANONICAL)
  lambda_3 = PLANCK / sqrt (2.0 * M_PI * (mass_molecular_3 / AVOGADRO) *
    BOLTZMANN * temp);
  mu_3_0   = 0.0;  /* 455.58e+03 / AVOGADRO ;  /* [J/particle] */
  mu_3     = mu_3_0 + R_GAS * temp * log (pressure * (1.0-RATIO_COMP1-
    RATIO_COMP2) / 1.0e+05) / AVOGADRO; /* [J/particle] */
  zz_3     = exp (mu_3/(BOLTZMANN * temp))/(lambda_3 * lambda_3 * lambda_3);
  /* temporary */
  /* zz_3     = beta * pressure * (1.0 - RATIO_COMP1 - RATIO_COMP2); */
#endif

#if defined (NANOTUBE)
  /* C - 3 */
  sigma_C_3 = (sigma_C_C + sigma_3_3) / 2;
  sigma2_C_3 = sigma_C_3 * sigma_C_3;
  sigma3_C_3 = sigma_C_3 * sigma_C_3 * sigma_C_3;

  epsilon_over_kb_C_3 = sqrt (epsilon_over_kb_C_C * epsilon_over_kb_3_3);
  eps_C_3 = epsilon_over_kb_C_3 * BOLTZMANN;
  eps4_C_3  = 4.0  * eps_C_3;
  eps48_C_3 = 48.0 * eps_C_3;
#endif
#endif  /* COMP3 */

/***  1 - 2  ***/
sigma_1_2 = (sigma_1_1 + sigma_2_2) / 2;
epsilon_over_kb_1_2 = sqrt (epsilon_over_kb_1_1 * epsilon_over_kb_2_2);

eps_1_2   = epsilon_over_kb_1_2 * BOLTZMANN;
eps4_1_2  = 4.0  * eps_1_2;
eps48_1_2 = 48.0 * eps_1_2;

sigma2_1_2 = sigma_1_2 * sigma_1_2;
sigma3_1_2 = sigma_1_2 * sigma_1_2 * sigma_1_2;

#if defined (COMP3)
  /***  1 - 3  ***/
  sigma_1_3 = (sigma_1_1 + sigma_3_3) / 2;
  epsilon_over_kb_1_3 = sqrt (epsilon_over_kb_1_1 * epsilon_over_kb_3_3);

  eps_1_3   = epsilon_over_kb_1_3 * BOLTZMANN;
  eps4_1_3  = 4.0  * eps_1_3;
  eps48_1_3 = 48.0 * eps_1_3;

  sigma2_1_3 = sigma_1_3 * sigma_1_3;
  sigma3_1_3 = sigma_1_3 * sigma_1_3 * sigma_1_3;

  /***  2 - 3  ***/
  sigma_2_3 = (sigma_2_2 + sigma_3_3) / 2;
  epsilon_over_kb_2_3 = sqrt (epsilon_over_kb_2_2 * epsilon_over_kb_3_3);

  eps_2_3   = epsilon_over_kb_2_3 * BOLTZMANN;
  eps4_2_3  = 4.0  * eps_2_3;
  eps48_2_3 = 48.0 * eps_2_3;

  sigma2_2_3 = sigma_2_3 * sigma_2_3;
  sigma3_2_3 = sigma_2_3 * sigma_2_3 * sigma_2_3;
#endif

#if defined (MOLECULAR_DYNAMICS)
  mass_particle_1 = mass_molecular_1 / AVOGADRO;
  mass_particle_2 = mass_molecular_2 / AVOGADRO;
  #if defined (COMP3)
    mass_particle_3 = mass_molecular_3 / AVOGADRO;
  #endif
#endif

#if defined (GRAND_CANONICAL) || defined (CANONICAL)
/* initialise output file */
out_current = fopen ("current.txt", "w");
```

```
/* write input data */
  fprintf (out_current, "Number of equilibration cycles:       %10d\n",   *Equil);
  fprintf (out_current, "Number of production cycles:          %10d\n",   *Prod);
  fprintf (out_current, "Sample frequency:                     %10d\n",   *Nsamp);
  fprintf (out_current, "Attempts to displ. part. / cycle:     %10d\n",   *Ndispl);
  fprintf (out_current, "Maximum displacement:                 %10.3e\n", *Dr);
  fprintf (out_current, "Cutoff distance:                      %10.3e\n",
      r_cutoff);
  #if defined (GRAND_CANONICAL)
  fprintf (out_current, "Attempts to exchange part. / cycle:   %10d\n",   *Nexch);
  #endif
  fprintf (out_current, "Number of particles:                  %10d\n",   npart);
  fprintf (out_current, "Temperature [K]:                      %10.3f\n", temp);
  fprintf (out_current, "Pressure [Pa]:                        %10.1f\n",
      pressure);
  #if defined (GRAND_CANONICAL) || defined (WIDOM)
  fprintf (out_current, "Chem.pot. ideal gas [J/particle]:     %10.5e\n",
      mu_ideal_gas);
  #endif
  fprintf (out_current, "Density particles 1 [particles/m3]:   %10.3e\n",
      rho_particles_1);
  fprintf (out_current, "Density particles 2 [particles/m3]:   %10.3e\n",
      rho_particles_2);
  #if defined (COMP3)
  fprintf (out_current, "Density particles 3 [particles/m3]:   %10.3e\n",
      rho_particles_3);
  #endif

  #if defined (CANONICAL) || defined (WIDOM)
    rho_mass = npart_comp [1] * mass_molecular_1 + npart_comp [2] *
      mass_molecular_2;
    pressure_actual = rho_particles_1 + rho_particles_2;
      #if defined (COMP3)
        rho_mass += npart_comp [3] * mass_molecular_3;
        pressure_actual += rho_particles_3;
      #endif
    rho_mass /= (AVOGADRO * tube_volume);
    pressure_actual *= R_GAS * TEMPERATURE_ACTUAL / AVOGADRO;
    fprintf (out_current, "Actual density - Begin [kg/m3]: %12.5f\n", rho_mass);
    fprintf (out_current, "Pressure (from actual particles) [Pa]: %12.1f\n",
      pressure_actual);
  #endif

  #if defined (LATTICE_CYLIND)
    fprintf (out_current, "Tube radius [m]:   %10.3e\n", tube_radius);
    fprintf (out_current, "Tube volume [m3]:  %10.3e\n", tube_volume);
  #elif defined (LATTICE_SQUARE)
    fprintf (out_current, "Box side   [m] :  %10.3e\n", box_side);
    fprintf (out_current, "Box volume [m3]:  %10.3e\n", tube_volume);
  #endif
  fprintf (out_current, "Model parameters:\n");
  fprintf (out_current, "    epsilon_1_1: %5.3e\n", eps_1_1);
  fprintf (out_current, "    sigma_1_1:   %5.3e\n", sigma_1_1);
  #if defined (NANOTUBE)
    fprintf (out_current, "    epsilon_C_1: %5.3e\n", eps_C_1);
    fprintf (out_current, "    sigma_C_1:   %5.3e\n", sigma_C_1);
  #endif
  #if defined (SHIFT_POTENT)
    /* calculate energy of the shift_potent (dirty trick) */
    energy_cutoff = 0.0;
    energy_function (&energy_cutoff, &vir, r_cutoff_2,
      sigma2_1_1, eps4_1_1, eps48_1_1, r_cutoff_2);
    fprintf (out_current, "Simulations with TRUNCATED AND SHIFTED potential:\n");
    fprintf (out_current, "  Potential truncated at: %10.3f\n", r_cutoff);
    fprintf (out_current, "  Energy shift:           %10.6f\n", energy_cutoff);
  #endif

  #if defined (TAIL_CORRECT)
    fprintf (out_current, "Simulations with tail correction:\n");
    fprintf (out_current, "  Potential truncated at Rc =%10.3f\n", r_cutoff);
    fprintf (out_current, "  Tail corrections:   energy = %10.3f  pressure =
%10.3f\n",
      correct_u(r_cutoff, rho, sigma3_1_1, eps4_1_1),
      correct_p(r_cutoff, rho, sigma3_1_1, eps4_1_1));
  #endif

  fclose (out_current);
  #endif /* GRAND_CANONICAL || CANONICAL */

  return;
}


/* write intermediate configuration to files */
#if defined (COMP3)
  void write_config_file (char simulation, int i_config, int i_cycl, double dr, int
comp1, int comp2, int comp3)
#else
  void write_config_file (char simulation, int i_config, int i_cycl, double dr, int
comp1, int comp2)
#endif
{
  char   i_config_str[10];
  char   file_interm_txt_str [50], file_interm_bin_str [50];
  int    out_bin_interm;
  FILE   *out_asc_interm;

  itoa (i_config, i_config_str, 10);

  if (simulation == 'c')
    {
      strcpy (file_interm_txt_str, "gcmc_config_interm_1_");
      strcpy (file_interm_bin_str, "gcmc_config_interm_1_");
    }
  else if (simulation == 'd')
    {
      strcpy (file_interm_txt_str, "md_config_interm_1_");
      strcpy (file_interm_bin_str, "md_config_interm_1_");
    }
  else
    exit (1);

  strcat (file_interm_txt_str, i_config_str);
  strcat (file_interm_txt_str, ".txt");
  out_asc_interm = fopen (file_interm_txt_str, "wt");
  store_asc (out_asc_interm, dr, i_cycl+1, comp1, npart_comp [1]);
```

```
fclose (out_asc_interm);

strcat (file_interm_bin_str, i_config_str);
strcat (file_interm_bin_str, ".bin");
out_bin_interm = open (file_interm_bin_str, O_WRONLY | O_CREAT | O_TRUNC |
    O_BINARY, S_IWRITE);
store_bin (out_bin_interm, dr, i_cycl+1, comp1);
close (out_bin_interm);

if (simulation == 'c')
  {
    strcpy (file_interm_txt_str, "gcmc_config_interm_2_");
    strcpy (file_interm_bin_str, "gcmc_config_interm_2_");
  }
else if (simulation == 'd')
  {
    strcpy (file_interm_txt_str, "md_config_interm_2_");
    strcpy (file_interm_bin_str, "md_config_interm_2_");
  }
else
  exit (1);

strcat (file_interm_txt_str, i_config_str);
strcat (file_interm_txt_str, ".txt");
out_asc_interm = fopen (file_interm_txt_str, "wt");
store_asc (out_asc_interm, dr, i_cycl+1, comp2, npart_comp [2]);
fclose (out_asc_interm);

strcat (file_interm_bin_str, i_config_str);
strcat (file_interm_bin_str, ".bin");
out_bin_interm = open (file_interm_bin_str, O_WRONLY | O_CREAT | O_TRUNC |
    O_BINARY, S_IWRITE);
store_bin (out_bin_interm, dr, i_cycl+1, comp2);
close (out_bin_interm);

#if defined (COMP3)
if (simulation == 'c')
  {
    strcpy (file_interm_txt_str, "gcmc_config_interm_3_");
    strcpy (file_interm_bin_str, "gcmc_config_interm_3_");
  }
else if (simulation == 'd')
  {
    strcpy (file_interm_txt_str, "md_config_interm_3_");
    strcpy (file_interm_bin_str, "md_config_interm_3_");
  }
else
  exit (1);

strcat (file_interm_txt_str, i_config_str);
strcat (file_interm_txt_str, ".txt");
out_asc_interm = fopen (file_interm_txt_str, "wt");
store_asc (out_asc_interm, dr, i_cycl+1, comp3, npart_comp [3]);
fclose (out_asc_interm);

strcat (file_interm_bin_str, i_config_str);
strcat (file_interm_bin_str, ".bin");
out_bin_interm = open (file_interm_bin_str, O_WRONLY | O_CREAT | O_TRUNC |
    O_BINARY, S_IWRITE);
  store_bin (out_bin_interm, dr, i_cycl+1, comp3);
  close (out_bin_interm);
  #endif
  return;
}


#if defined (MOLECULAR_DYNAMICS)

/* count particles outside tube */
void count_outside (int *count_radius, int *count_out_low, int *count_low, int
*count_high, int *count_out_high)
{
  int  i;

  *count_radius = 0;
  *count_low    = 0;
  *count_high   = 0;
  *count_out_low  = 0;
  *count_out_high = 0;

  for (i = 0; i < npart; i++)
    {
      if ((x[i]*x[i]+y[i]*y[i]) <= tube_radius_2)
        (*count_radius)++;
      if (z[i] < 0.0)
        (*count_out_low)++;
      if (z[i] > tube_height)
        (*count_out_high)++;
      if (z[i] < (tube_height / 3))
        (*count_low)++;
      if (z[i] > (2.0*tube_height/3))
        (*count_high)++;
    }
  return;
}

/* initial velocity distribution */
void init_velocity (void)
{
  double sum_v_x, sum_v_y, sum_v_z;
  double sum_v2;
  double fs;
  int i;

  v_x = (double *) malloc (npart * sizeof (double));
  v_y = (double *) malloc (npart * sizeof (double));
  v_z = (double *) malloc (npart * sizeof (double));

  f_x = (double *) malloc (npart * sizeof (double));
  f_y = (double *) malloc (npart * sizeof (double));
  f_z = (double *) malloc (npart * sizeof (double));

  sum_v_x = 0.0;
  sum_v_y = 0.0;
  sum_v_z = 0.0;
  sum_v2  = 0.0;

  for (i = 0; i < npart; i++)
```

```
    {
      v_x [i] = 0.5 - random1000 ();
      v_y [i] = 0.5 - random1000 ();
      v_z [i] = 0.5 - random1000 ();

      sum_v_x += v_x [i];
      sum_v_y += v_y [i];
      sum_v_z += v_z [i];
      sum_v2  += mass [i] * (v_x [i] * v_x [i] + v_y [i] * v_y [i] + v_z [i] * v_z
      [i]);
    }

  /* velocity center of mass */
  sum_v_x /= npart;
  sum_v_y /= npart;
  sum_v_z /= npart;

  /* scale factor of the velocities */
  fs = sqrt (3.0 * npart * BOLTZMANN * temp / sum_v2);

  /* set desired kinetic energy and set velocity center of mass to zero */
  /* position previous time step */
  for (i = 0; i < npart; i++)
    {
      v_x [i]    = (v_x [i] - sum_v_x) * fs;
      v_y [i]    = (v_y [i] - sum_v_y) * fs;
      v_z [i]    = (v_z [i] - sum_v_z) * fs;
      x_prev [i] = x [i] - v_x [i] * dt;
      y_prev [i] = y [i] - v_y [i] * dt;
      z_prev [i] = z [i] - v_z [i] * dt;
    }
  return;
}


/* rescale velocities */
double rescale_velocity (void)
{
  double x_new, y_new, z_new, temp_new;
  double sum_v2;
  double factor;
  int flag_factor;
  int i;

  flag_factor = 0;
  sum_v2  = 0.0;
  for (i = 0; i < npart; i++)
    sum_v2  += mass [i] * (v_x [i] * v_x [i] + v_y [i] * v_y [i] + v_z [i] * v_z
      [i]);
  temp_new = (sum_v2 / (3.0 * npart)) / BOLTZMANN;

  factor = sqrt (temp / temp_new);

  while ((factor < 0.99) || (factor > 1.01))
    {
      flag_factor = 1;
      for (i = 0; i < npart; i++)
        {
          v_x [i] *= factor;
          v_y [i] *= factor;
          v_z [i] *= factor;
        }
      sum_v2  = 0.0;
      for (i = 0; i < npart; i++)
        sum_v2  += mass [i] * (v_x [i] * v_x [i] + v_y [i] * v_y [i] + v_z [i] *
      v_z [i]);

      temp_new = (sum_v2 / (3.0 * npart)) / BOLTZMANN;
      factor = sqrt (temp / temp_new);
    }

  if (flag_factor)
    {
      for (i = 0; i < npart; i++)
        {
          x_new = x_prev [i] + dt * v_x [i];
          y_new = y_prev [i] + dt * v_y [i];
          z_new = z_prev [i] + dt * v_z [i];
          if (check_space (x_new, y_new, z_new, i))
            {
              x [i] = x_new;
              y [i] = y_new;
              z [i] = z_new;
            }
          else
            {
              x [i] = x_prev [i];
              y [i] = y_prev [i];
              z [i] = z_prev [i];
              x_prev [i] = x_prev_prev [i];
              y_prev [i] = y_prev_prev [i];
              z_prev [i] = z_prev_prev [i];
            }
        }
    }
  return (temp_new);
}

void calculate_forces (void)
{
  int    i, j;
  double ff_x, ff_y, ff_z;
  #if defined (NANOTUBE)
    double ff_wall_x, ff_wall_y, ff_wall_z;
  #endif

  for (i = 0; i < npart; i++)
    {
      f_x [i] = 0.0;
      f_y [i] = 0.0;
      f_z [i] = 0.0;
    }

  for (i = 0; i < npart-1; i++)
    {
      /* other particles */
      for (j = i+1; j < npart; j++)
        {
          /* forces exerted by j on i */
          force_particles_function (i, j, &ff_x, &ff_y, &ff_z, r_cutoff_2);

          /* updated force */
          f_x [i] += ff_x; /* add force exerted by j on i */
          f_x [j] -= ff_x; /* add force exerted by i on j */

          f_y [i] += ff_y;
          f_y [j] -= ff_y;

          f_z [i] += ff_z;
          f_z [j] -= ff_z;
        }

      #if defined (NANOTUBE)
      /* wall particles */
      j = 0;
      while ( (z_carbon [j] < (z[i] - r_cutoff_wall)) && (j < n_carbon) )
        j++;

      while ( (z_carbon [j] < (z[i] + r_cutoff_wall)) && (j < n_carbon) )
        {
          /* forces exerted by j on i */
          force_wall_function (i, j, &ff_wall_x, &ff_wall_y, &ff_wall_z);

          /* updated force */
          f_x [i] += ff_wall_x;  /* add force exerted by j on i */
          f_y [i] += ff_wall_y;
          f_z [i] += ff_wall_z;

          j++;
        }
      #endif
    }
  return;
}

/* integrate equations of motion */

/* apparently, the following has to be avoided
   the initial configuration is an equilibrium, that results from the previous
      Monte-Carlo simulation
   the sudden displacement of a particle may accidentaly create a completely
      unphysical situation
   (e.g. two particles too close together)
*/

double integrate_motion (FILE *out)
{
  double  sum_v_x, sum_v_y, sum_v_z;
  double  sum_v_x_abs, sum_v_y_abs, sum_v_z_abs;
  double  x_new, y_new, z_new, z_new_temp;
  double  sum_v2;
  double  energy_kinetic;
  double  displac2, displac2_max;
  int     i;
  int     flag_fill_particle;

  double  x_int_1, y_int_1, x_int_2, y_int_2, x_int, y_int;
  double  dist_inside, dist_outside, dist_total;
  double  d1_square, d2_square;
  double  x_final_1, x_final_2, y_final_1, y_final_2;
  double  v_x_new, v_y_new, v_z_new;

  sum_v_x = 0.0;
  sum_v_y = 0.0;
  sum_v_z = 0.0;
  sum_v_x_abs = 0.0;
  sum_v_y_abs = 0.0;
  sum_v_z_abs = 0.0;
  sum_v2      = 0.0;
  displac2_max = 0.0;

  for (i = 0; i < npart; i++)
    {
      flag_fill_particle = 0;

      /* Verlet algorithm */
      x_new = 2.0 * x [i] - x_prev [i] + dt2 * f_x [i] / mass [i];
      y_new = 2.0 * y [i] - y_prev [i] + dt2 * f_y [i] / mass [i];
      z_new = 2.0 * z [i] - z_prev [i] + dt2 * f_z [i] / mass [i];
      /*
      if (z_new > (2.0*tube_height/3.0))
        z_new_temp = z_new - 1.0*tube_height/3.0;
      else if (z_new < (tube_height/3.0))
        z_new_temp = z_new + 1.0*tube_height/3.0;
      else
      */
      z_new_temp = z_new;


      /* velocity */
      v_x_new = (x_new - x_prev [i]) / (two_dt);
      v_y_new = (y_new - y_prev [i]) / (two_dt);
      v_z_new = (z_new - z_prev [i]) / (two_dt);

      #if defined (CONFINED_PARTICLES)
      /* If a particle has crossed the wall:                              */
      /*   1. Calculate new position                                      */
      /*   2. Calculate new (fictive) previous position                   */
      /*   3. Calculate new velocity v_x, v_y (same value, changed direction) */
      if ((x_new*x_new + y_new*y_new) > tube_radius_2)
        {
          if ((x_new*x_new + y_new*y_new) > (1.2 * tube_radius_2))
            {
              fill_particle (i);
              npart--;
              flag_fill_particle = 1;
            }
          else
            {
              /* total distance that has to be travelled */
              dist_total = sqrt ((x_new-x[i]) * (x_new-x[i]) + (y_new-y[i]) *
            (y_new-y[i]));

              /* intersection of wall and trajectory: x_int, y_int */
              /* wall: (x-0)**2 + (y-0)**2 = tube_radius_2         */
              /* trajectory: (y-y[i])/(x-x[i])=(yy-y[i])/(xx-x[i]) */
```

```
solve_circle_line (0.0, 0.0, tube_radius, x[i], y[i], (y_new-
    y[i])/(x_new-x[i]), &x_int_1, &y_int_1, &x_int_2, &y_int_2);

/* distance between intersection and outside point */
d1_square = (x_int_1 - x_new) * (x_int_1 - x_new) + (y_int_1 - y_new)
    * (y_int_1 - y_new);
d2_square = (x_int_2 - x_new) * (x_int_2 - x_new) + (y_int_2 - y_new)
    * (y_int_2 - y_new);
if (d1_square < d2_square)
  {
    x_int = x_int_1;
    y_int = y_int_1;
    dist_outside = sqrt (d1_square);
    /* distance travelled outside wall */
  }
else
  {
    x_int = x_int_2;
    y_int = y_int_2;
    dist_outside = sqrt (d2_square);
    /* distance travelled outside wall */
  }
dist_inside = dist_total - dist_outside;

/* final point:
   intersection between circle (x_int, y_int, dist_outside)
   and line (y-yy)/(x-xx) = (y_int - 0)/(x_int - 0),
   within tube_wall
*/
solve_circle_line (x_int, y_int, dist_outside, x_new, y_new,
    y_int/x_int,
  &x_final_1, &y_final_1, &x_final_2, &y_final_2);

if ((x_final_1*x_final_1 + y_final_1*y_final_1) <= tube_radius_2)
  {
    x_new = x_final_1;
    y_new = y_final_1;
  }
else
  {
    x_new = x_final_2;
    y_new = y_final_2;
  }

if ((x_new*x_new+y_new*y_new) > tube_radius_2)
  {
    printf ("Particle outside tube radius\n");
    printf ("i:%5d  time:%14.5e\n", i, time);
    printf ("x_final_1: %14.5e  y_final_1: %14.5e\n", x_final_1,
    y_final_1);
    printf ("x_final_2: %14.5e  y_final_2: %14.5e\n", x_final_2,
    y_final_2);
    printf ("Press any key to exit program.\n");
    getch ();
    exit (1);
  }

/* let's see, if the new position will be acceptable */
```

```
        /* if the new position is acceptable, the fictive previous position
      will be calculated */
        /* in principle, the fictive previous position should be outside the
      tube */
        if (check_space (x_new, y_new, z_new_temp, i))
          {
            /* fictive previous position:
               intersection between circle (x_int, y_int, dist_inside)
               and line (y-y[i])/(x-x[i]) = (y_int - 0)/(x_int - 0),
               outside of tube wall
            */
            solve_circle_line (x_int, y_int, dist_inside, x[i], y[i],
                  y_int/x_int,
              &x_final_1, &y_final_1, &x_final_2, &y_final_2);

            if ((x_final_1*x_final_1 + y_final_1*y_final_1) >= tube_radius_2)
              {
                x[i] = x_final_1;
                y[i] = y_final_1;
              }
            else if ((x_final_2*x_final_2 + y_final_2*y_final_2) >=
                  tube_radius_2)
              {
                x[i] = x_final_2;
                y[i] = y_final_2;
              }
            else
              {
                printf ("Particle inside tube radius instead of outside\n");
                printf ("i:%5d  time:%14.5e\n", i, time);
                printf ("x_final_1: %14.5e  y_final_1: %14.5e\n", x_final_1,
            y_final_1);
                printf ("x_final_2: %14.5e  y_final_2: %14.5e\n", x_final_2,
            y_final_2);
                printf ("Press any key to exit program.\n");
                getch ();
                exit (1);
              }
          }
        /* modified velocity direction: in principle, not necessary */
      }
  }

/*
if ((zz > (tube_height + tube_radius)) || (zz < -tube_radius))
  {
    fill_particle (i);
    npart--;
    flag_fill_particle = 1;
  }
*/
/*
if (zz > (6.0*tube_height/7.0))
  {
    zz   -= 5.0*tube_height/7.0;
    z[i] -= 5.0*tube_height/7.0;
  }
else if (zz < (tube_height/7.0))
  {
    zz   += 5.0*tube_height/7.0;
    z[i] += 5.0*tube_height/7.0;
  }
*/
/*
if (zz > tube_height)
  {
    dist_total   = zz - z[i];
    dist_outside = zz - tube_height;
    zz = tube_height - dist_outside;
    z [i] = zz + dist_total;
    v_z [i] *= -1.0;
  }
else if (zz < 0.0)
  {
    dist_total   = z [i] - zz;
    dist_outside = -1.0 * zz;
    zz = dist_outside;
    z [i] = zz - dist_total;
    v_z [i] *= -1.0;
  }
*/
#endif

if (flag_fill_particle == 0)
  {
    if (check_space (x_new, y_new, z_new_temp, i))
      {
        /*
        if (z_new > (2.0*tube_height/3.0))
          {
            z_new -= 1.0*tube_height/3.0;
            z[i]  -= 1.0*tube_height/3.0;
          }
        else if (z_new < (tube_height/3.0))
          {
            z_new += 1.0*tube_height/3.0;
            z[i]  += 1.0*tube_height/3.0;
          }
        */

        /* update positions previous time */
        x_prev_prev [i] = x_prev [i];
        y_prev_prev [i] = y_prev [i];
        z_prev_prev [i] = z_prev [i];
        x_prev      [i] = x [i];
        y_prev      [i] = y [i];
        z_prev      [i] = z [i];
        x           [i] = x_new;
        y           [i] = y_new;
        z           [i] = z_new;
        /*
        displac2 = (x[i] - x_prev[i])*(x[i] - x_prev[i])
                 + (y[i] - y_prev[i])*(y[i] - y_prev[i])
                 + (z[i] - z_prev[i])*(z[i] - z_prev[i]);
        if (displac2 > displac2_max)
          displac2_max = displac2;
        */
        v_x         [i] = v_x_new;
```

```
              v_y          [i] = v_y_new;
              v_z          [i] = v_z_new;
            }

          /* total kinetic energy */
          sum_v2 += mass [i] * (v_x [i] * v_x [i] + v_y [i] * v_y [i] + v_z [i] *
      v_z [i]);
        }

      /* velocity center of mass */
      sum_v_x     += v_x [i];
      sum_v_y     += v_y [i];
      sum_v_z     += v_z [i];
      sum_v_x_abs += fabs (v_x [i]);
      sum_v_y_abs += fabs (v_y [i]);
      sum_v_z_abs += fabs (v_z [i]);
    } /* i=1,npart */

  order_particles (x, y, z, npart);

  /* check sum_v_x, sum_v_y, sum_v_z */
  fprintf (out, "%10.4f%10.4f%10.4f\n", sum_v_x/sum_v_x_abs, sum_v_y/sum_v_y_abs,
      sum_v_z/sum_v_z_abs);

  /* instantaneous temperature */
  /* reference temperature should not be modified */
  /* temp = (sum_v2 / (3.0 * npart)) / BOLTZMANN;  */

  /* total energy */
  energy_kinetic = 0.5 * sum_v2;
  /* printf ("displac_max: %14.5e\n", sqrt(displac2_max)); */

  return (energy_kinetic);
}


/* self-diffusivities in nanotube axial direction */
void self_diffusivities (double time_aver_diffus)
{
  int     i;
  FILE    *out_diffus;
  int     n_diffus_1, n_diffus_2;
  double  diffus_1, diffus_2;
  #if defined (COMP3)
    int     n_diffus_3;
    double  diffus_3;
  #endif

  out_diffus = fopen ("diffus.txt", "a+");

  diffus_1   = 0.0;
  n_diffus_1 = 0;
  diffus_2   = 0.0;
  n_diffus_2 = 0;
  #if defined (COMP3)
    diffus_3   = 0.0;
    n_diffus_3 = 0;
  #endif
  for (i = 0; i < npart; i++)
    {
      if (
        /*
        (z_initial [i] > tube_height/3)
        && (z_initial [i] < (2.0 * tube_height/3))
        */
        /* (z[i] > 0.0) && (z[i] < tube_height) */
        /* (z[i] > (tube_height/7.0)) && (z[i] < (6.0*tube_height/7.0)) */
        (z[i] > (tube_height/3.0)) && (z[i] < (2.0*tube_height/3.0))
        )
        {
          switch (comp [i])
            {
              case 1:
                diffus_1 += (z[i] - z_initial [i]) * (z[i] - z_initial [i]);
                n_diffus_1++;
                break;
              case 2:
                diffus_2 += (z[i] - z_initial [i]) * (z[i] - z_initial [i]);
                n_diffus_2++;
                break;
              #if defined (COMP3)
              case 3:
                diffus_3 += (z[i] - z_initial [i]) * (z[i] - z_initial [i]);
                n_diffus_3++;
                break;
              #endif
            }
        }
    }
  fprintf (out_diffus, "\nTime:%14.5e\n", time_aver_diffus);
  diffus_1 /= (2.0 * time_aver_diffus * n_diffus_1);
  fprintf (out_diffus, "Comp.: 1  Part.: %3d  Ds:%14.5e", n_diffus_1, diffus_1);
  printf ("%14.5e", diffus_1);

  if (n_diffus_2 > 0)
    {
      diffus_2 /= (2.0 * time_aver_diffus * n_diffus_2);
      fprintf (out_diffus, " Comp.: 2 Part.: %3d  Ds:%14.5e", n_diffus_2,
            diffus_2);
      printf ("%14.5e", diffus_2);
    }

  #if defined (COMP3)
  if (n_diffus_3 > 0)
    {
      diffus_3 /= (2.0 * time_aver_diffus * n_diffus_3);
      fprintf (out_diffus, " Comp.: 3   Part.: %3d  Ds:%14.5e", n_diffus_3,
            diffus_3);
      printf ("%14.5e", diffus_3);
    }
  #endif
  printf ("\n");
  fprintf (out_diffus, "\n");
  fclose (out_diffus);
  return;
}
#endif /* MOLECULAR_DYNAMICS */
```

```
/*********************************  m a i n  ********************************/

void main (void)
{
  /*****  VARIABLE DECLARATIONS  - begin  *****/
  char  component_1, component_2;
  #if defined (COMP3)
    char  component_3;
  #endif

  #if defined (GRAND_CANONICAL) || defined (CANONICAL)
    double  energy_running, energy_current, virial_running, virial_current, dr;
    int     equil, prod, nsamp, ii, n_displ, move_attempt, move_success,
            i_cycl_mc, n_cycl, n_moves, i_move, nsamp_av;
    double  rho_particles_1_av, rho_particles_2_av;
    #if defined (COMP3)
      double rho_particles_3_av;
    #endif
  #endif

  #if defined (MOLECULAR_DYNAMICS)
    int     i_cycl_md, count_out;
    double  time_aver_diffus;
    int     i, j, count_radius, count_low, count_high, count_out_low,
      count_out_high;
    double  energy_potential, energy_kinetic;
    double  virial;
    /* variable defined, so that energy_potential_function may be called */
    double  temp_new;
  #endif

  #if defined (WRITE_CONFIG)
    int  i_config;
  #endif
  #if defined (GRAND_CANONICAL)
    int     n_exch, exch_success, exch_attempt;
    double  ran;
  #endif
  #if defined (NANOTUBE)
    int     i_c;
    double  z_min, z_max;
  #endif

  #if defined (WIDOM)
    double  wtest_1, wtest_2, mu_1_excess, mu_2_excess;
    double  widom_inst_1, widom_inst_2;
    int     nsamp_av_widom_1, nsamp_av_widom_2;
    int     i_widom, i_widom_av;
    #if defined (COMP3)
      double  wtest_3, mu_3_excess;
      double  widom_inst_3;
      int     nsamp_av_widom_3;
    #endif
  #endif
```

```
/* output files for storing configuration */
FILE   *out_current;

/*****  VARIABLE DECLARATIONS  - end  *****/

/*****  read coordinates of atoms in CNT  *****/
#if defined (NANOTUBE)
  /* n_carbon = get_c_coordinates_bin (); */
  n_carbon = get_c_coordinates_txt ();
  order_carbon_atoms (x_carbon, y_carbon, z_carbon, n_carbon);
  for (i_c = 0; i_c < n_carbon; i_c++)
    printf ("%5d%12.5e%12.5e%12.5e\n", i_c, *(x_carbon+i_c), *(y_carbon+i_c),
    *(z_carbon+i_c));
  printf ("\n");

  /* check minimum z */
  z_min = 0.0;
  z_max = 0.0;
  for (i_c = 0; i_c < n_carbon; i_c++)
    {
      if (z_carbon [i_c] < z_min)
        z_min = z_carbon [i_c];
      if (z_carbon [i_c] > z_max)
        z_max = z_carbon [i_c];
    }
#endif

/* initialize system */
srand (1);
component_1 = COMP1;
component_2 = COMP2;
#if defined (COMP3)
  component_3 = COMP3;
#endif

/**********  calculate various constants  **********/
#if defined (GRAND_CANONICAL)
  #if defined (COMP3)
    set_data (&equil, &prod, &nsamp, &n_displ, &dr, &n_exch, component_1,
            component_2, component_3);
  #else
    set_data (&equil, &prod, &nsamp, &n_displ, &dr, &n_exch, component_1,
            component_2);
  #endif
#elif defined (CANONICAL)
  #if defined (COMP3)
    set_data (&equil, &prod, &nsamp, &n_displ, &dr, component_1, component_2,
    component_3);
  #else
    set_data (&equil, &prod, &nsamp, &n_displ, &dr, component_1, component_2);
  #endif
#elif defined (MOLECULAR_DYNAMICS)
  #if defined (COMP3)
    set_data (component_1, component_2, component_3);
  #else
    set_data (component_1, component_2);
  #endif
#endif
```

```
/******************************  MONTE-CARLO  *****************************/
#if defined (GRAND_CANONICAL)  || defined (CANONICAL)

/* print initial configuration */
#if defined (WRITE_CONFIG)
  i_cycl_mc = 0;
  i_config = 0;
  #if defined (COMP3)
    write_config_file ('c', i_config, i_cycl_mc, dr, 1, 2, 3);
  #else
    write_config_file ('c', i_config, i_cycl_mc, dr, 1, 2);
  #endif
  i_config++;
#endif

#if defined (GRAND_CANONICAL)
  n_moves = n_displ + n_exch;
#else
  n_moves = n_displ;
#endif

/* total energy of the system */
energy_running = energy_potential_function (&virial_running);
printf ("Total energy initial configuration: %12.5e\n", energy_running);
printf ("Total virial initial configuration: %12.5e\n", virial_running);

out_current = fopen ("current.txt", "a+");
fprintf (out_current, "Total energy initial configuration: %12.5e\n",
    energy_running);
fprintf (out_current, "Total virial initial configuration: %12.5e\n",
    virial_running);
fclose (out_current);

/***  start MC cycle  ***/
for (ii = 1; ii <= 2; ii++)
  {
    /* ii=1: equilibration cycles */
    /* ii=2: production cycles    */
    if (ii == 1)
      {
        n_cycl = equil;
        if (n_cycl != 0)
          printf ("Start equilibration\n");
      }
    else if (n_cycl != 0)
      {
        n_cycl = prod;
        printf ("Start production\n");
      }

    move_attempt = 0;
    move_success = 0;
    #if defined (GRAND_CANONICAL)
      exch_attempt = 0;
      exch_success = 0;
    #endif

    /* averaging within equilibration or production cycle */
    rho_particles_1_av = 0.0;
```

```
rho_particles_2_av = 0.0;
#if defined (WIDOM)
  wtest_1 = 0.0;
  wtest_2 = 0.0;
#endif
#if defined (COMP3)
  rho_particles_3_av = 0.0;
  #if defined (WIDOM)
    wtest_3 = 0.0;
  #endif
#endif
nsamp_av = 0;
#if defined (WIDOM)
  nsamp_av_widom_1 = 0;
  nsamp_av_widom_2 = 0;
  #if defined (COMP3)
    nsamp_av_widom_3 = 0;
  #endif
#endif


/* initialize the subroutine that adjusts the maximum displacement */
adjust (move_attempt, move_success, &dr, 0);

for (i_cycl_mc = 0; i_cycl_mc < n_cycl; i_cycl_mc++)
/* ncycl = equil or prod */
  {
    if ((i_cycl_mc % 10) == 0)
      printf ("%5d\n", i_cycl_mc);
    for (i_move = 0; i_move < n_moves; i_move++)
      {
        /***  core part of MC cycle  ***/
        #if defined (GRAND_CANONICAL)
            ran = random1000 () * n_moves;
            if (ran < n_displ)
              /* attempt to displace a particle */
              mcmove (&energy_running, &virial_running, &move_attempt,
                  &move_success, dr);
            else
              /* attempt to exchange a particle with the bath */
              mcexch (&energy_running, &virial_running, &exch_attempt,
                  &exch_success);
        #else
            mcmove (&energy_running, &virial_running, &move_attempt,
                  &move_success, dr);
        #endif
      }

    /* if production cycle, average results */
    /*
    if (ii == 2)
      { */
    /* however, for the time being, results are also averaged during
      equilibration */
    /* sample averages */
    if ( ((i_cycl_mc+1) % nsamp) == 0)
      {
        /* sample ((i_cycl_mc+1), energy, virial);  let's leave this
            subroutine for later */
        /* to determine excess chemical potential */
        nsamp_av++;
        rho_particles_1_av = rho_particles_1_av + ((double)npart_comp [1]) /
      tube_volume;
        rho_particles_2_av = rho_particles_2_av + ((double)npart_comp [2]) /
      tube_volume;
        #if defined (COMP3)
          rho_particles_3_av = rho_particles_3_av + ((double)npart_comp [3])
      / tube_volume;
        #endif
        #if defined (WIDOM)
          widom_inst_1 = widom (1);
          nsamp_av_widom_1++;
          wtest_1 += exp (-beta * widom_inst_1);

          widom_inst_2 = widom (2);
          nsamp_av_widom_2++;
          wtest_2 += exp (-beta * widom_inst_2);

          #if defined (COMP3)
          widom_inst_3 = widom (3);
          nsamp_av_widom_3++;
          wtest_3 += exp (-beta * widom_inst_3);
          #endif
       #endif
      }
    /*  } */

    /* periodically write configuration to file and adjust maximum
      displacement */
    if (((i_cycl_mc+1) % (2000)) == 0)
      {
        out_current = fopen ("current.txt", "a+");
        energy_current = energy_potential_function (&virial_current);

        /***  output to screen  ***/
        #if defined (COMP3)
          printf ("\n======>> Done %5d out of %5d %8d %8d %8d %8d\n",
            i_cycl_mc+1, n_cycl, npart, npart_comp [1], npart_comp [2],
      npart_comp [3]);
        #else
          printf ("\n======>> Done %5d out of %5d %8d %8d %8d\n",
            i_cycl_mc+1, n_cycl, npart, npart_comp [1], npart_comp [2]);
        #endif
        printf ("Total energy at end of simulation: %12.5e\n",
            energy_current);
        printf ("   running energy             : %12.5e\n", energy_running);
        printf ("   difference                 : %12.5e\n", energy_current -
      energy_running);
        printf ("Total virial at end of simulation: %12.5e\n",
            virial_current);
        printf ("   running virial             : %12.5e\n", virial_running);
        printf ("   difference                 : %12.5e\n\n", virial_current
      - virial_running);
        /* printf ("Press any key to continue ...\n"); */
        /* getch(); */

        /***  output to file  ***/
        #if defined (COMP3)
```

```
        fprintf (out_current, "\n======>> Done %5d out of %5d %8d %8d %8d
    %8d\n",
          i_cycl_mc+1, n_cycl, npart, npart_comp [1], npart_comp [2],
    npart_comp [3]);
      #else
        fprintf (out_current, "\n======>> Done %5d out of %5d %8d %8d
    %8d\n",
          i_cycl_mc+1, n_cycl, npart, npart_comp [1], npart_comp [2]);
      #endif
      fprintf (out_current, "Total energy at end of simulation: %12.5e\n",
          energy_current);
      fprintf (out_current, "   running energy             : %12.5e\n",
          energy_running);
      fprintf (out_current, "   difference                 : %12.5e\n",
          energy_current - energy_running);
      fprintf (out_current, "Total virial at end of simulation: %12.5e\n",
          virial_current);
      fprintf (out_current, "   running virial             : %12.5e\n",
          virial_running);
      fprintf (out_current, "   difference                 : %12.5e\n\n",
          virial_current - virial_running);

      /* up-to-date average of excess chemical potential */
      #if defined (WIDOM)
        fprintf (out_current, "Up-to-date average of excess chemical
          potential\n");
        printf  ("Up-to-date average of excess chemical potential\n");
        mu_1_excess = -log (wtest_1/nsamp_av_widom_1)/beta;
        fprintf (out_current, "  mu_1_excess: %12.5e\n", mu_1_excess);
        printf  ("  mu_1_excess: %12.5e\n", mu_1_excess);
        mu_2_excess = -log (wtest_2/nsamp_av_widom_2)/beta;
        fprintf (out_current, "  mu_2_excess: %12.5e\n", mu_2_excess);
        printf  ("  mu_2_excess: %12.5e\n", mu_2_excess);
          #if defined (COMP3)
            mu_3_excess = -log (wtest_3/nsamp_av_widom_3)/beta;
            fprintf (out_current, "  mu_3_excess: %12.5e\n", mu_3_excess);
            printf  ("  mu_3_excess: %12.5e\n", mu_3_excess);
          #endif
        #endif
      fclose  (out_current);


      /***  write intermediate configuration to files  ***/
      #if defined (WRITE_CONFIG)
        #if defined (COMP3)
          write_config_file ('c', i_config, i_cycl_mc, dr, 1, 2, 3);
        #else
          write_config_file ('c', i_config, i_cycl_mc, dr, 1, 2);
        #endif
        i_config++;
      #endif

      /* adjust maximum displacements */
      adjust (move_attempt, move_success, &dr, i_cycl_mc);
    }
}

/* conclusion of either equilibration or production cycles */
out_current = fopen ("current.txt", "a+");
```

```
if (move_attempt != 0)
  {
    printf ("\nNumber of attempts to displace a particle: %10d\n",
      move_attempt);
    printf ("  success: %10d  '(=' %5.2f '%)'\n", move_success,
      100.0* (double)(move_success)/(double)(move_attempt));
    fprintf (out_current, "\nNumber of attempts to displace a particle:
      %10d\n", move_attempt);
    fprintf (out_current, "  success: %10d  '(=' %5.2f '%)'\n", move_success,
      100.0* (double)(move_success)/(double)(move_attempt));
  }

#if defined (GRAND_CANONICAL)
if (exch_attempt != 0)
  {
    printf ("\nNumber of attempts to exchange particles: %10d\n",
      exch_attempt);
    printf ("  success: %10d  '(=' %5.2f '%)'\n", exch_success,
      100.0* (double)(exch_success)/(double)(exch_attempt));
    fprintf (out_current, "\nNumber of attempts to exchange particles:
      %10d\n", exch_attempt);
    fprintf (out_current, "  success: %10d  '(=' %5.2f '%)'\n", exch_success,
      100.0* (double)(exch_success)/(double)(exch_attempt));
  }
#endif

/* Widom's test particle method with fixed configuration */
/* as many test particles of each component will be generated,
   as there are particles in the system */
#if defined (WIDOM)
  wtest_1 = 0.0;
  wtest_2 = 0.0;
  #if defined (COMP3)
    wtest_3 = 0.0;
  #endif
  for (i_widom_av = 0; i_widom_av < 10; i_widom_av++)
    {
      for (i_widom = 0; i_widom < 100*npart; i_widom++)
        {
          wtest_1 += exp (-beta * widom (1));
          wtest_2 += exp (-beta * widom (2));

          #if defined (COMP3)
          wtest_3 += exp (-beta * widom (3));
          #endif
        }

      fprintf (out_current, "Excess chemical potential with fixed
            configuration (%6d tests)\n", (i_widom_av+1)*100*npart);
      printf  ("Excess chemical potential with fixed configuration (%6d
            tests)\n", (i_widom_av+1)*100*npart);

      mu_1_excess = -log (wtest_1/((i_widom_av+1)*100*npart))/beta;
      fprintf (out_current, "  mu_1_excess: %12.5e\n", mu_1_excess);
      printf  ("  mu_1_excess: %12.5e\n", mu_1_excess);

      mu_2_excess = -log (wtest_2/((i_widom_av+1)*100*npart))/beta;
      fprintf (out_current, "  mu_2_excess: %12.5e\n", mu_2_excess);
      printf  ("  mu_2_excess: %12.5e\n", mu_2_excess);
      #if defined (COMP3)
        mu_3_excess = -log (wtest_3/((i_widom_av+1)*100*npart))/beta;
        fprintf (out_current, "  mu_3_excess: %12.5e\n", mu_3_excess);
        printf  ("  mu_3_excess: %12.5e\n", mu_3_excess);
      #endif
    }
#endif

/*  test total energy  */
/*  energy and virial are gradually modified during the simulation     */
/*  energy_total and virial_total are calculated at the end            */
/*  in principle, at the end, energy_total = en and virial_total = vir */
energy_current = energy_potential_function (&virial_current);
if (fabs ((energy_current - energy_running)/energy_current) > 0.01)
  {
    printf ("LARGE DIFFERENCE BETWEEN energy_current AND energy_running\n");
    printf ("Energy current: %12.5e\n", energy_current);
    printf ("Energy running: %12.5e\n", energy_running);
    fprintf (out_current, "LARGE DIFFERENCE BETWEEN energy_current AND
      energy_running\n");
    fprintf (out_current, "Energy current: %12.5e\n", energy_current);
    fprintf (out_current, "Energy running: %12.5e\n", energy_running);
  }
if (fabs ((virial_current - virial_running)/virial_current) > 0.01)
  {
    printf ("LARGE DIFFERENCE BETWEEN virial_current AND virial_running\n");
    printf ("Virial current: %12.5e\n", virial_current);
    printf ("Virial running: %12.5e\n", virial_running);
    fprintf (out_current, "LARGE DIFFERENCE BETWEEN virial_current AND
      virial_running\n");
    fprintf (out_current, "Virial current: %12.5e\n", virial_current);
    fprintf (out_current, "Virial running: %12.5e\n", virial_running);
  }

printf ("Total energy at end of simulation: %12.5e\n", energy_current);
printf ("   running energy  : %12.5e\n", energy_running);
printf ("   difference      : %12.5e\n", energy_current - energy_running);
printf ("Total virial at end of simulation: %12.5e\n", virial_current);
printf ("   running virial  : %12.5e\n", virial_running);
printf ("   difference      : %12.5e\n\n", virial_current - virial_running);
fprintf (out_current, "Total energy at end of simulation: %12.5e\n",
      energy_current);
fprintf (out_current, "   running energy  : %12.5e\n", energy_running);
fprintf (out_current, "   difference      : %12.5e\n", energy_current -
      energy_running);
fprintf (out_current, "Total virial at end of simulation: %12.5e\n",
      virial_current);
fprintf (out_current, "   running virial  : %12.5e\n", virial_running);
fprintf (out_current, "   difference      : %12.5e\n\n", virial_current -
      virial_running);

/* determine excess chemical potential reservoir */
if ((rho_particles_1_av != 0.0) && (nsamp_av != 0))
  {
    rho_particles_1_av /= nsamp_av;
    fprintf (out_current, "Average density 1 [particles/m3]: %12.5e\n",
      rho_particles_1_av);
```

```
        rho_particles_2_av /= nsamp_av;
        fprintf (out_current, "Average density 2 [particles/m3]: %12.5e\n",
          rho_particles_2_av);

        #if defined (COMP3)
          rho_particles_3_av /= nsamp_av;
          fprintf (out_current, "Average density 3 [particles/m3]: %12.5e\n",
          rho_particles_3_av);
        #endif

        rho_mass =  rho_particles_1_av * mass_molecular_1 / AVOGADRO;
        rho_mass += rho_particles_2_av * mass_molecular_2 / AVOGADRO;
        #if defined (COMP3)
          rho_mass += rho_particles_3_av * mass_molecular_3 / AVOGADRO;
        #endif
        fprintf (out_current, "Actual density - End [kg/m3]: %12.5f\n",
          rho_mass);

      }
    fclose (out_current);
  }
/***  end MC cycle  ***/

/***  write final configuration to files  ***/
#if defined (COMP3)
  write_config_file ('c', i_config, i_cycl_mc, dr, 1, 2, 3);
#else
  write_config_file ('c', i_config, i_cycl_mc, dr, 1, 2);
#endif
#endif
/*******************************  MONTE-CARLO  ******************************/

/***************************  MOLECULAR DYNAMICS  ***************************/
#if defined (MOLECULAR_DYNAMICS)
  out_current = fopen ("current_MD.txt", "w");
  fprintf (out_current, "%12s%8s%12s%7s", "time", "temp", "energy", "count");
  fprintf (out_current, "%10s%10s%10s\n", "sum_v_x", "sum_v_y", "sum_v_z");
  fprintf (out_current, "%10s%10s%10s%10s%10s\n",
    "(height)", "(radius)", "(ratio)", "(ratio)", "(ratio)");

  /* initiate time and cycle counter */
  i_cycl_md = 0;
  time      = 0.0;
  i_config  = 0;

  /***  read configuration from file, sort particles  ***/
  #if defined (COMP3)
    npart = read_config_files (npart_comp, 1,
      "config_1.txt", 2, "config_2.txt", 3, "config_3.txt");
  #else
    npart = read_config_files (npart_comp, 1,
      "config_1.txt", 2, "config_2.txt");
  #endif

  for (i = 0; i < npart; i++)
    {
      /* z [i] = z [i] + tube_height / 7.0; */
      z [i] = z [i] + tube_height / 3.0;
    }
```

```
/**  add additional particles  ***/
/*
for (i = 0; i < npart; i++)
  {
    x    [i + npart] = x    [i];
    y    [i + npart] = y    [i];
    z    [i + npart] = z    [i] + tube_height/7;
    comp [i + npart] = comp [i];
    mass [i + npart] = mass [i];
  }

for (i = 0; i < npart; i++)
  {
    x    [i + 2*npart] = x    [i];
    y    [i + 2*npart] = y    [i];
    z    [i + 2*npart] = z    [i] + 2.0*tube_height/7;
    comp [i + 2*npart] = comp [i];
    mass [i + 2*npart] = mass [i];
  }

for (i = 0; i < npart; i++)
  {
    x    [i + 3*npart] = x    [i];
    y    [i + 3*npart] = y    [i];
    z    [i + 3*npart] = z    [i] + 3.0*tube_height/7;
    comp [i + 3*npart] = comp [i];
    mass [i + 3*npart] = mass [i];
  }

for (i = 0; i < npart; i++)
  {
    x    [i + 4*npart] = x    [i];
    y    [i + 4*npart] = y    [i];
    z    [i + 4*npart] = z    [i] + 4.0*tube_height/7;
    comp [i + 4*npart] = comp [i];
    mass [i + 4*npart] = mass [i];
  }
npart *= 5;
*/

x_prev      = (double *) malloc (npart * sizeof (double));
y_prev      = (double *) malloc (npart * sizeof (double));
z_prev      = (double *) malloc (npart * sizeof (double));
x_prev_prev = (double *) malloc (npart * sizeof (double));
y_prev_prev = (double *) malloc (npart * sizeof (double));
z_prev_prev = (double *) malloc (npart * sizeof (double));
/* record initial axial coordinate of each particle */
z_initial = (double *) malloc (npart * sizeof (double));

order_particles (x,y,z,npart);

/* initial velocity distribution */
init_velocity ();

/***************************  MD Equilibration  ***************************/
fprintf (out_current, "\n\nEquilibration\n\n");
for (i = 0; i < npart; i++)
  z_initial[i] = z[i];
```

```
while (time <= TIME_EQUIL)
  {
    calculate_forces ();
    energy_kinetic   = integrate_motion (out_current);
    energy_potential = energy_potential_function (&virial);
    temp_new = rescale_velocity ();

    count_outside (&count_radius, &count_out_low, &count_low, &count_high,
        &count_out_high);
    if ((i_cycl_md % 2) == 0)
      printf ("time:%12.4e cycle:%5d low:%4d %4d high:%4d %4d ins.radius:%4d
        temp:%8.2f\n",
      time, i_cycl_md, count_out_low, count_low, count_high, count_out_high,
        count_radius, temp_new);

    time += dt;
    i_cycl_md++;
    #if defined (WRITE_CONFIG)
    if (((i_cycl_md+1) % (1000)) == 0)
      {
        #if defined (COMP3)
          write_config_file ('d', i_config, i_cycl_md, 0.0, 1, 2, 3);
        #else
          write_config_file ('d', i_config, i_cycl_md, 0.0, 1, 2);
        #endif
        i_config++;
      }
    #endif

    fprintf (out_current, "%12.5e%8.2f%14.5e%7d%7d%7d",
      time, temp_new, energy_kinetic + energy_potential, count_low, count_high,
        count_radius);
  }

/* count particles from central region that have fallen out */
/* CHECK */
count_out = 0;
for (i = 0; i < npart; i++)
  {
    if (
         (z_initial [i] > tube_height/3)
      && (z_initial [i] < (2.0 * tube_height/3))
      && ((z[i] < 0.0) || (z[i] > tube_height))
    )
    count_out++;
  }
fprintf (out_current, "\nOutside particles from center: %5d\n", count_out);

/*****************************  MD Averaging  *****************************/
fprintf (out_current, "\n\nAveraging\n\n");
for (i = 0; i < npart; i++)
  z_initial[i] = z[i];
time_aver_diffus = time + DT_AVER_DIFFUS;

while (time <= (TIME_EQUIL+TIME_AVER))
  {
    calculate_forces ();
    energy_kinetic   = integrate_motion (out_current);
    energy_potential = energy_potential_function (&virial);
    temp_new = rescale_velocity ();

    count_outside (&count_radius, &count_out_low, &count_low, &count_high,
        &count_out_high);
    if ((i_cycl_md % 2) == 0)
      printf ("time:%12.4e cycle:%5d low:%4d %4d high:%4d %4d ins.radius:%4d
        temp:%8.2f\n",
        time, i_cycl_md, count_out_low, count_low, count_high, count_out_high,
        count_radius, temp_new);

    time += dt;
    i_cycl_md++;
    #if defined (WRITE_CONFIG)
    if (((i_cycl_md+1) % (1000)) == 0)
      {
        #if defined (COMP3)
          write_config_file ('d', i_config, i_cycl_md, 0.0, 1, 2, 3);
        #else
          write_config_file ('d', i_config, i_cycl_md, 0.0, 1, 2);
        #endif
        i_config++;
      }
    #endif

    fprintf (out_current, "%12.5e%8.2f%14.5e%7d%7d%7d",
      time, temp_new, energy_kinetic + energy_potential, count_low, count_high,
        count_radius);
    if (time >= time_aver_diffus)
      {
        self_diffusivities (time_aver_diffus);
        time_aver_diffus += DT_AVER_DIFFUS;
      }
  }

/* count particles */
fprintf (out_current, "\nTotal particles: %5d\n", npart);

/* count particles from central region that have fallen out */
/* CHECK */
count_out = 0;
for (i = 0; i < npart; i++)
  {
    if (
         (z_initial [i] > tube_height/3)
      && (z_initial [i] < (2.0 * tube_height/3))
      && ((z[i] < 0.0) || (z[i] > tube_height))
    )
    count_out++;
  }
fprintf (out_current, "\nOutside particles from center: %5d\n", count_out);

/***  write final configuration to files  ***/
#if defined (COMP3)
  write_config_file ('d', i_config, i_cycl_md, 0.0, 1, 2, 3);
#else
  write_config_file ('d', i_config, i_cycl_md, 0.0, 1, 2);
#endif
```

```
    free (x_prev);
    free (y_prev);
    free (z_prev);
    free (x_prev_prev);
    free (y_prev_prev);
    free (z_prev_prev);
    free (z_initial);
    free (v_x);
    free (v_y);
    free (v_z);
    free (f_x);
    free (f_y);
    free (f_z);
    free (mass);
  #endif

  free (x);
  free (y);
  free (z);
  free (comp);

  #if defined (NANOTUBE)
    free (x_carbon);
    free (y_carbon);
    free (z_carbon);
  #endif

  return;
}
```

```
/***  Nanotube.c  ***/

/* Atomic coordinates for a (n,m) nanotube in the unit cell */

/* Reference: Physical Properties of Carbon Nanotubes
              Imperial College Press
              by R.Saito, G.Dresselhaus and M.S.Dresselhaus
*/

/* Input: n_tube, m_tube */
/* Output:
   tube_xyz.txt:  coordinates
   en_xyz2.bin:   work file for program A-3
   tube_parameters.txt
*/

#include  <fcntl.h>
#include  <sys\stat.h>
#include  <math.h>
#include  <stdio.h>
#include  <stdlib.h>
#include  <io.h>
#include  <conio.h>

#define  ACC      1.42   /* distance between C atoms */
#define  NK       2000
#define  N_TUBE   10     /* tube parameter n */
#define  M_TUBE   10     /* tube parameter m */
#define  PI       3.1415926525
#define  EPSILON  0.00001


/* the modulus is the remainder after arithmetic division */
int modulus (int i, int j)
{
  int ratio, result;

  ratio = 1;
  while ((ratio * j) <= i)
    ratio++;
  ratio--;
  result = i - (ratio * j);
  if (result != (i%j))
    {
      printf ("Function modulus wrongly programmed\n");
      printf ("Press any key...\n");
      getch ();
      exit (1);
    }
  return (result);
}


/* calculate radial coordinates */
void get_radial (double x, double y, double *r, double *theta)
{
  *r = sqrt (x*x + y*y);
  /* x = 0 */
  if ((x > -EPSILON) && (x < EPSILON))
```

```
    {
      if (y > 0.0)
        *theta = PI/2;
      else
        *theta = 3.0 * PI / 2.0;
    }
  /* y = 0 */
  else if ((y > -EPSILON) && (y < EPSILON))
    {
      if (x > 0.0)
        *theta = 0.0;
      else
        *theta = PI;
    }
  else if (x > 0.0)
    {
      if (y > 0.0)
        *theta = atan (y/x);
      else
        *theta = 3.0 * PI / 2.0 + atan (x/-y);
    }
  else
    {
      if (y > 0.0)
        *theta = PI / 2.0 + atan (-x/y);
      else
        *theta = PI + atan (-y/-x);
    }
  return;
}


/* highest common divisor */
int igcm (int ii, int jj)
{
  int i, j, iw, ir;

  i = abs (ii);
  j = abs (jj);

  /* find highest number and name it i */
  if (j>i)
    {
      iw = j;
      j = i;
      i = iw;
    }
  if (j == 0)
    {
      return (i);
    }

  /* now i > j and j != 0 */
  while (1)
    {
      ir = modulus (i, j);
      /* check modulus function */
      if (ir != i%j)
        {
          printf ("Something wrong in function modulus\n");
          printf ("Press any key...");
          getch ();
          exit (1);
        }

      if (ir == 0)
        return (j);
      else
        {
          i = j;
          j = ir;
        }
    }
}


void gen11 (int n, int m, int *np, int *nq, int *ndr, FILE *out)
{
  int    nnp[100], nnq[100];
  double a;
  long   l2;
  double l, nt, nt2, dt;
  int    nd, nr, ns, nn, n60;
  int    ichk, i, j1, j2;
  int    inp, inq;

  /* nd: greatest common divisor of n and m */
  nd = igcm (n, m);
  fprintf (out, "d: %3d\n", nd);

  /* ndr: greatest common divisor of (2m+n) and (2n+m) - eq. (3.7) */
  if (!modulus (n-m, 3*(*nq)))
    *ndr = 3 * nd;
  else
    *ndr = nd;
  fprintf (out, "d_r: %3d\n", *ndr);

  /* lattice constant */
  a = sqrt (3.0) * ACC;
  fprintf (out, "a: %6.2f\n", a);

  l2 = (n * n) + (m * m) + (n * m);
  if (l2 <= 0)
    {
      printf ("l2 <= 0");
      printf ("Press any key ...");
      getch ();
      exit (1);
    }

  /* L/a */
  l = sqrt ((double) (l2)) + EPSILON;
  fprintf (out, "L/a: %6.2f\n", l);

  /* nanotube diameter - eq. (3.2) */
  dt = a * sqrt ((double) (l2)) / PI;
  fprintf (out, "d_t: %6.2f\n", dt);
```

```
/* T: translational vector - eqs. (3.6), (3.8) */
/* t1 */
nr =   (2 * m + n) / (*ndr);
/* t2 */
ns = - (2 * n + m) / (*ndr);
fprintf (out, "T = (%2d,%2d)\n", nr, ns);

/* |T| - eq. (3.8) */
nt2 = 3 * a * a * l2/((*ndr)*(*ndr));
nt = sqrt ((double) (nt2)) + EPSILON;

/* |T|/a */
fprintf (out, "|T|/a: %6.2f\n", nt/a);

/* N - eq. (3.9) */
nn = 2 * l2/(*ndr);
fprintf (out, "N:%3d\n", nn);

/* R */
/* The symmetry vector R is used for generating the coordinates
    of carbon atoms in the nanotube */
ichk = 0;
if (nr == 0)
  n60 = 1;
else
  n60 = nr;

for (inp = -abs (n60); inp <= abs (n60); inp++)
  {
    for (inq = -abs (ns); inq <= abs (ns); inq++)
      {
        j2 = nr * inq - ns * inp;
        if (j2 == 1)
          {
            j1 = m * inp - n * inq;
            if ((j1 > 0) && (j1 < nn))
              {
                ichk++;
                nnp [ichk] = inp;
                nnq [ichk] = inq;
                *np = inp;
                *nq = inq;
              }
          }
      }
  }

if (!ichk)
  {
    printf ("not found p, q strange !");
    printf ("Press any key ...");
    getch ();
    exit (1);
  }

if (ichk >= 2)
  {
    printf ("more than 1 pair of p,q strange !");
    printf ("Press any key ...");
```

```
      getch ();
      exit (1);
    }

  if ((nr != 0) && (ns != 0))
    {
      i = 1;
      while ((i <= ichk) && ((m * nnp [i] - n * nnq [i]) >= nn))
        i++;
      *np = nnp [1];
      *nq = nnq [1];
      return;
    }
}


/***  main  ***/
void main (void)
{
  double  x[NK], y[NK], z[NK];
  double  acc;
  int     np, nq, ndr;
  double  sq3, a;
  double  r, c, t, rs, tube_diam;
  int     n, m, nk, nn, nn2, mm;
  double  q1, q2, q3, q4, q5;
  double  h1, h2;
  int     i, ii;
  double  x1, y1, z1, z3;
  double  x2, y2, z2;
  double  radial_r, radial_theta;
  int     k, kk, kk2;
  FILE    *out_tube, *out_en, *out_check;
  int     out_bin;

  n   = N_TUBE;
  m   = M_TUBE;
  nk  = NK;
  acc = ACC;
  sq3 = sqrt (3.0);

  out_check = fopen ("tube_parameters.txt", "wt");
  fprintf (out_check, "n: %3d\n", n);
  fprintf (out_check, "m: %3d\n", m);

  gen11 (n, m, &np, &nq, &ndr, out_check);

  /* symmetry vector */
  fprintf (out_check, "R: (%2d,%2d)\n", np, nq);

  /* a: length of the unit vector */
  a = sq3 * acc;

  /* |R| */
  r = a * sqrt ((double) (np * np + nq * nq + np * nq));

  /* |C_h| = L */
  c = a * sqrt ((double) (n * n + m * m + n * m));
```

```
/* tube diameter */
tube_diam = c / PI;
fprintf (out_check, "tube diameter =%8.4f\n", tube_diam);

/* |T| */
t = sq3 * c / (double)ndr;
fprintf (out_check, "|T|= %f\n", t);

/* nn: number of hexagons in the unit cell N */
/* eq. (3.9) */
nn = 2 * (n*n + m*m + m*n)/ndr;
fprintf (out_check, "N: %2d\n", nn);

/* M */
mm = n * np - m * nq;
fprintf (out_check, "M:%3d\n", mm);

/* rs: tube radius */
if ((2 * nn) > nk)
  {
    printf ("Parameter nk is too small\n");
    printf ("Press any key ...");
    getch ();
    exit (1);
  }

/* L/2PI */
rs = c / (2.0 * PI);

/* q1: chiral angle for C_h    */
/* q2: "chiral" angle for R    */
/* q3: angle between C_h and R */

q1 = atan ((sq3 * (double) (m)) /(double) (2 * n  + m));
q2 = atan ((sq3 * (double) (nq))/(double) (2 * np + nq));
q3 = q1 - q2;

/* q4: period of an angle for A atom */
/* q5: difference of the angle between A and B atoms */

q4 = 2.0 * PI / (double) (nn);
q5 = acc * cos ((PI/6.0)-q1)/c * 2.0 * PI;

/* h2: Delta z between the A and B atoms */
h1 = fabs (t) / fabs (sin (q3));
h2 = acc * sin ((PI / 6.0) - q1);

fprintf (out_check, "q1 = %6.2f\n", q1 * 180.0 / PI);
fprintf (out_check, "q2 = %6.2f\n", q2 * 180.0 / PI);
fprintf (out_check, "q4 = %6.2f\n", q4 * 180.0 / PI);
fprintf (out_check, "q5 = %6.2f\n", q5 * 180.0 / PI);

/* Calculate 2 * nn atoms in the unit cell */

/* A atom */
ii = 0;
for (i = 0; i < nn; i++)
  {
    k = (int) ((double) (i) * fabs (r)/h1);
```

```
      x1 = rs * cos ((double) (i) * q4);
      y1 = rs * sin ((double) (i) * q4);
      z1 = (double) (((double) (i) * fabs (r) - (double) (k) * h1)) * sin (q3);
      kk2 = abs ((int) (z1/t)) + 1;

      /*  check if the A atom is in the unit cell 0 < z1 < t */
      if (z1 > (t - 0.02))
        z1 = z1 - t * (double) (kk2);
      if (z1 < -0.02)
        z1 = z1 + t * (double) (kk2);
      ii++;
      x [ii] = x1;
      y [ii] = y1;
      z [ii] = z1;

      /* B atoms */
      z3 = ((double) (i) * fabs (r) - (double) (k) * h1) * sin(q3) - h2;
      ii++;

      /* check if the B atom is in the unit cell 0 < z3 < t */
      if ((z3 >= -0.02) && (z3 <= (t - 0.02)))
        /* yes */
        {
          x2 = rs * cos ((double) (i) * q4 + q5);
          y2 = rs * sin ((double) (i) * q4 + q5);
          z2 = (double) ((double) (i) * fabs (r) - (double) (k) * h1)
            * sin (q3) - h2;
          x [ii] = x2;
          y [ii] = y2;
          z [ii] = z2;
        }
      else
        /* no */
        {
          x2 = rs * cos ((double) (i) * q4 + q5);
          y2 = rs * sin ((double) (i) * q4 + q5);
          z2 = (double) ((double) (i) * fabs (r) - (double) (k+1) * h1)
            * sin (q3) - h2;
          kk = abs ((int) (z2/t)) + 1;
          if (z2 > (t - 0.01))
            z2 = z2 - t * (double) (kk);
          if (z2 < (t - 0.01))
            z2 = z2 + t * (double) (kk);
          x [ii] = x2;
          y [ii] = y2;
          z [ii] = z2;
        }
    }

  /* Output to the file tube_xyz.txt */
  out_tube = fopen ("tube_xyz.txt", "wt");
  fprintf (out_tube, "%d \n", 2 * nn);
  fprintf (out_tube, "%12s %10s %10s %10s %10s %10s\n",
        "x", "y", "z", "r", "theta", "theta/PI");
  for (i = 1; i <= nn * 2; i++)
    {
      get_radial (x[i], y[i], &radial_r, &radial_theta);
      fprintf (out_tube, "C %10.5f %10.5f %10.5f %10.5f %10.5f %10.5f\n",
        x[i], y[i], z[i], radial_r, radial_theta, radial_theta/PI);
    }
  fclose (out_tube);

  /* Output to the file en_xyz2.txt */
  out_en = fopen ("en_xyz2.txt", "wt");
  fprintf (out_en, "%5d\n", 2 * nn);
  fprintf (out_en, "%25.5f %25.5f\n", t, acc);
  for (i = 1; i <= nn * 2; i++)
    fprintf (out_en, "%5d %25.20f %25.20f %25.20f\n", i, x[i], y[i], z[i]);
  fclose (out_en);
  fclose (out_check);

  /* Output to the binary file en_xyz2.bin */
  out_bin = open ("en_xyz2.bin", O_WRONLY | O_CREAT | O_TRUNC | O_BINARY,
      S_IWRITE);

  nn2 = 2 * nn;
  write (out_bin, &nn2, sizeof (int));
  write (out_bin, &t,   sizeof (double));
  write (out_bin, &acc, sizeof (double));

  for (i = 1; i <= nn * 2; i++)
    {
      write (out_bin, &i,      sizeof (int));
      write (out_bin, &(x[i]), sizeof (double));
      write (out_bin, &(y[i]), sizeof (double));
      write (out_bin, &(z[i]), sizeof (double));
    }
  close (out_bin);

  return;
}
```