5. Toteuta alla mainitut funktiot yksinkertaista sukupuuta mallintavaan ohjelmaan. Funktiokommenteissa on annettu avuksi myös malli, jonka pohjalta voit tehdä toteutuksen (tehtäväpohjan orjallinen noudattaminen ei ole välttämätöntä, mutta person-tietorakennetta tai annettuja funktiorajapintoja ei saa muuttaa). Voit olettaa että muistin varaus onnistuu aina.

a) **newPerson**, joka luo uuden henkilön. Henkilöllä on parametrina annettu nimi, sekä (mahdollisesti) annetut vanhemmat. Mikäli vanhempi ei ole tiedossa, kyseisen parametrin arvo on NULL. Aluksi henkilöllä ei ole lapsia. Kuten tehtäväpohja ehdottaa, on suositeltavaa luoda erillinen funktio vanhempien lapsien päivitystä varten. 'children' on dynaamisesti skaalautuva taulukko, jossa on **nChildren** alkiota. Kukin taulukon alkio on **osoitin person-tietueeseen** (4 p)

b) **printDescendants**, joka tulostaa henkilön nimen, sekä kaikkien henkilön jälkeläisten nimet (lapset, lapsenlapset, lapsenlapsenlapset, ...). Saman henkilön nimi saa tulostua toistuvasti, ja tulostusjärjestyksellä ei ole väliä. (2p)

```
struct person {
	char name[30];
	struct person *father;
	struct person *mother;
	unsigned int nChildren; // number of children
	struct person **children;  // dynamic array of children pointers
};

struct person *newPerson(const char *name,
		struct person *dad, struct person *mom)
{
	// allocate memory;

	// initialize fields based on parameters.
	// Pay attention to avoiding overflowing the reserved fields
	// (number of children is initially 0)

	// add new child for mom and dad (if they are not NULL)
	// (separate function, such as 'addChild' recommended)

	// return pointer to newly allocated person
}

void printDescendants(struct person *p)
{
	// print name of person p

	// iterate through children and print their names
	// go through their children, etc. (recursion recommended)
}
```