# Computation 5EIA0
# Exercise 7: Graphs: Keep it in the Family (v1.5 October 14, 2020)
# Deadline Wednesday 21 October 23:55

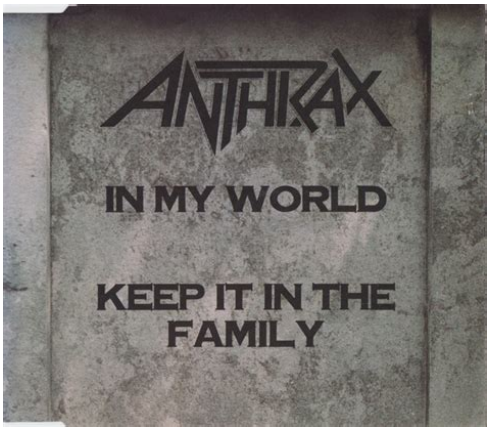


Figure 1: Keep it in the family by Anthrax.
`https://www.youtube.com/watch?v=p8yipLe71Yo`

In this assignment you will keep track of family trees and check ancestry between people. Every person has name and birth year. Additionally each person has a biological mother and a father, each of which has a mother and father, and so on. An example family tree is shown in Figure 2. We will use this family tree as the running example.

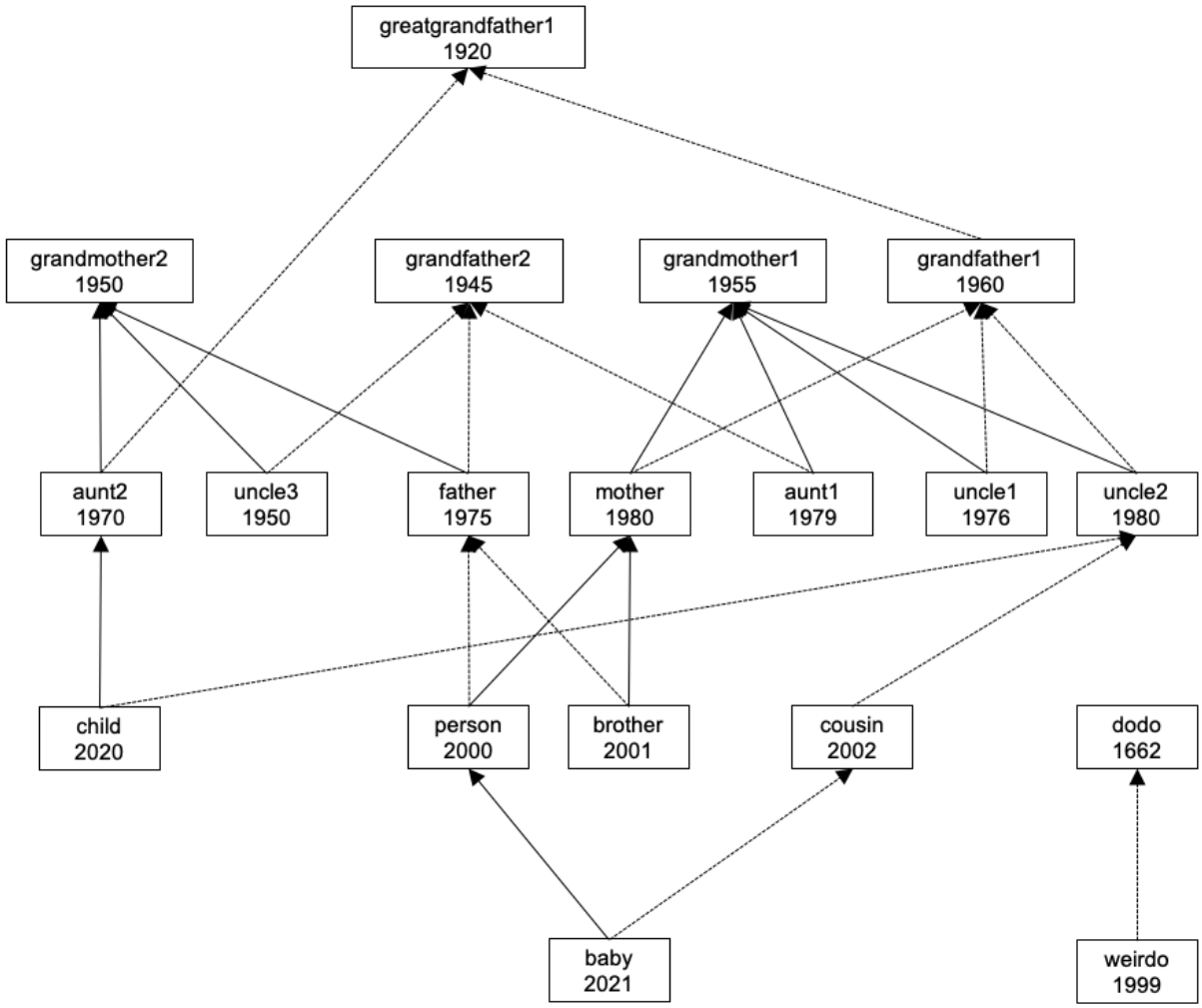


Figure 2: Our running example of family trees. Solid lines indicate mothers, dotted lines fathers.

The family tree in Figure 2 illustrates that persons may be related to different degrees:

- *direct ancestor*, e.g. child-parent or child-great grandparent
- indirectly by having a *common ancestor*, e.g. two children that have the same mother, or baby and uncle1 that have a common (great grand) parents grandmother1 and grandfather1.
- no common ancestor but with *descendants in common*, e.g. father and mother or grandmother2 and grandfather1.
- no ancestors and descendants in common, eg. baby and weirdo.

You will implement a number of different functions:

- print a family tree
- insert a person
- add parents of a person
- check if a person is a direct ancestor of another person (e.g. parent or grandparent)
- delete all persons born before a certain year
- print a list of all common ancestors of two persons

Ancestor functions are a good example of recursive functions: person A is an ancestor of person B if A is the mother or father of B, or if A is an ancestor of B's mother or an ancestor of B's father. We shall look in more detail at these below.

**Task 1**. As usual we have a main program that accepts commands. Start by implementing a main program that prints the prompt and implements the quit command. An example output (where the user pressed the return key, then entered q followed by return) is:

```
Command? q
Bye!
```

**Task 2**. Next we need to define the data structure that we will use in this exercise. A person has a name (`char *name`) and a birth year (`int birthyear`). Thus we need to define a `struct person` datatype that contains both elements. Furthermore each person has a mother and father, i.e. a pointer to a person (`struct person *mother, struct person *father`). Because not all persons are ancestors of a single person (such as child or baby or weirdo), we need to also keep a linked list of all persons. For this reason we need a final `struct person *nextperson` member in the structure. Define a `struct person *families` variable in the main function with initial value NULL.

**Task 3**. Implement the 'i' command to add a person and the `void addperson (struct person **families, char *name, int birthyear)` function that inserts a new person at the start of the linked list. There may be no two people with the same name (even if they have different birth years). It is not allowed to use the name "unknown". The name of a person does not contain spaces and may be read with the `%s` format string. Set the mother and father members of the new person to NULL.

The addperson function can give two error messages:

```
A person cannot be called unknown.
Person NAME already exists.
```

NAME should of course be equal to the duplicate person's name. In both cases the linked list is unchanged.

As part of the addperson function you must therefore be able to check if a person is in the list already. Make a separate function `struct person *findperson (struct person *families, char *person)` for this, since you will need it more often in the remainder.

```
Command? i
Name? person
Birth year? 2000
Command? i
Name? mother
Birth year? 1980
Command? i
Name? grandfather1
Birth year? 1960
Command? i
Name? greatgrandfather1
Birth year? 1920
Command? q
Bye!
```

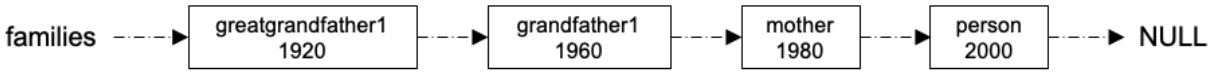


Figure 3: The linked list after inserting greatgrandfather1, grandfather1, mother, and person. Note that the mother and father fields are NULL.

**Task 4**. Add the 't' command and the `void print (struct person *families)` function that prints out the persons as they are ordered in the list in the following format.

```
Command? i
Name? greatgrandfather1
Birth year? 1920
Command? i
Name? grandmother1
Birth year? 1955
Command? t
name=grandmother1 birthyear=1955 mother=unknown father=unknown
name=greatgrandfather1 birthyear=1920 mother=unknown father=unknown
Command? q
Bye!
```

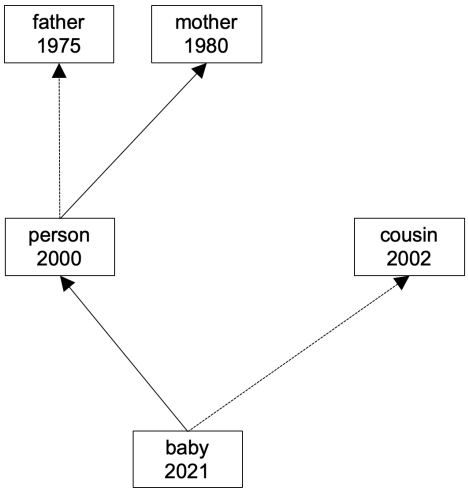


Figure 4: The family tree used in Task 5 and Task 7.

**Task 5**. Now add the 'p' command to insert/add the parents of a person with the `void addparents (struct person *families, char *person, char *mother, char *father)` function. (The mother and/or father members of the person are updated only; the linked list is unchanged.) If the name of the mother is `unknown` then no mother is added (and similarly for the father). An error message must be given if the mother or father cannot be found in the list of persons (`Person NAME not found.`), or if the person already has a father or mother (`Person NAME already has a mother NAME.`).

```
Command? i
Name? baby
Birth year? 2021
Command? i
Name? person
Birth year? 2000
Command? i
Name? cousin
Birth year? 2002
Command? i
Name? mother
Birth year? 1980
Command? i
Name? father
Birth year? 1975
Command? p
Person? PrinsPils
Mother? who
Father? unknown
Person PrinsPils not found.
Command? p
Person? baby
Mother? who
Father? unknown
Person who not found.
Command? p
Person? baby
Mother? person
Father? cousin
Command? p
Person? baby
Mother? person
Father? unknown
Person baby already has a mother person.
Command? p
Person? person
Mother? mother
Father? father
name=father birthyear=1975 mother=unknown father=unknown
name=mother birthyear=1980 mother=unknown father=unknown
name=cousin birthyear=2002 mother=unknown father=unknown
name=person birthyear=2000 mother=mother father=father
name=baby birthyear=2021 mother=person father=cousin
Command? q
Bye!
```

**Task 6**. The next command 'a' (for ancestor) checks if a person is a direct ancester of another person. Add the function: `int ancestor (struct person *young, struct person *old, int level)`. The old person is an ancestor of the young person if

- old is equal to young, or
- old is an ancestor of the mother of young, or
- old is an ancestor of the father of young

The function should

- return -1 when old is not an ancestor of young,
- return 0 when young equals old,
- return 1 when old is a parent of young,
- return 2 when old is a grandparent of young,
- return 3 when old is a great grandparent of young,
- return 4 when old is a great great grandparent of young, etc.

You can write this in a recursive function
`int ancestor (struct person *young, struct person *old, int level)`

1. return -1 when young or old are NULL (i.e. you're at a top of the family tree and you cannot go up)
2. return 0 when young equals old
3. check if old is an ancestor of young's mother (call this m)
4. check if old is an ancestor of young's father (call this f)
5. if m == -1 or f == -1 then return max(m,f) else return min(m,f)

Call the function in the main program with level equal to 0 and increase it when you recursively call ancestor. The minimum in the last bullet ensures that if old is both (e.g.) a grandparent and a parent then we return the closest relationship (i.e. parent). The function need not be longer than 7 lines of code!

```
Command? a
Person? grandmother1
Possible ancestor? baby
baby is not an ancestor of grandmother1.
```

The integer that your function returns should be converted a nice printed string, such as:

```
brother is not an ancestor of baby.
greatgrandfather1 is the same person as greatgrandfather1.
greatgrandfather1 is a parent of grandfather1.
greatgrandfather1 is a grandparent of mother.
greatgrandfather1 is a great grandparent of person.
greatgrandfather1 is a great great grandparent of baby.
p11 is a great great great great great great great grandparent of p1.
```

Your function should work for any number of "great"s.

A way to visualise how the recursion works is shown in Figure 5. The recursion follows the red line from person to grandfather, as shown in Figure 6

1. young.name != old.name
2. young.m.name != old.name
3. young.m.m.name != old.name
4. young.m.m.m == NULL
5. young.m.m.f == NULL
6. young.m.f.name != old.name
7. young.m.f.m == NULL
8. young.m.f.f.name != old.name
9. young.m.f.f.m == NULL
10. young.m.f.f.f == NULL
11. young.f.name != old.name
12. young.f.m.name != old.name
13. young.f.m.m == NULL
14. young.f.m.f == NULL
15. young.f.f.name == old.name

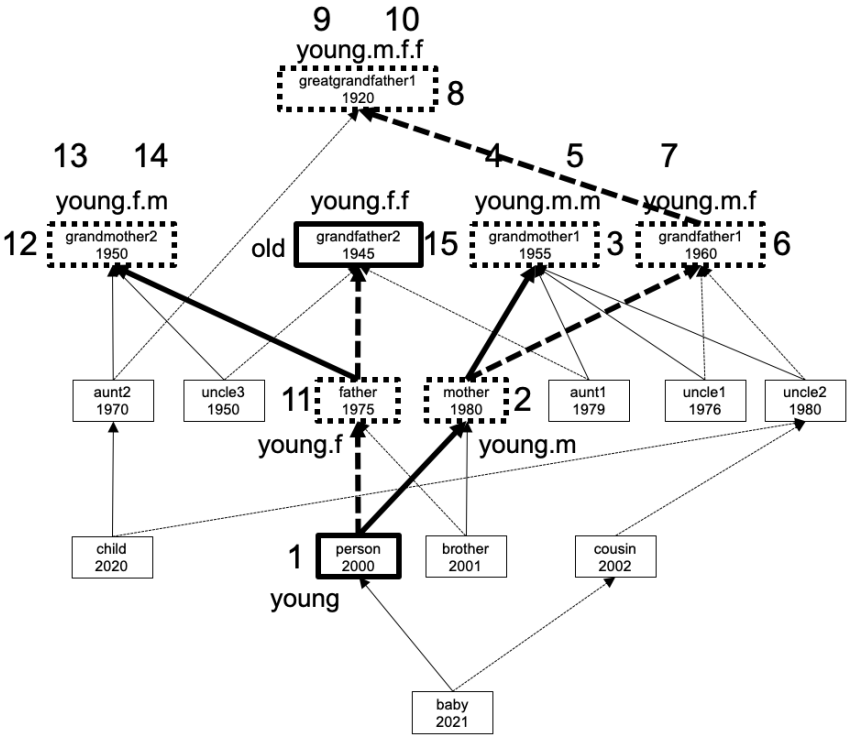


Figure 5: Visualising the ancestor recursion.

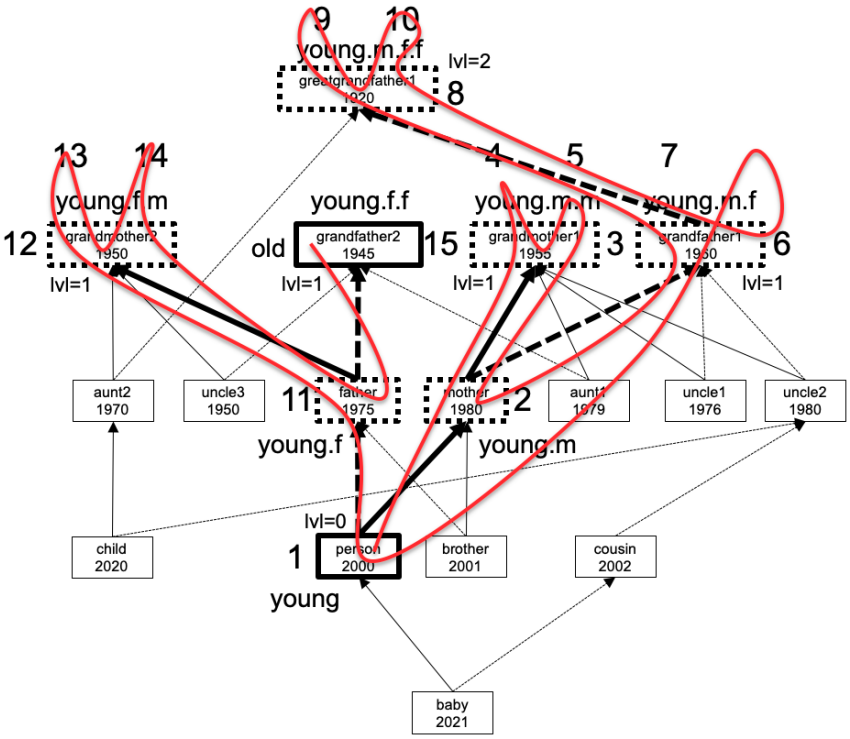


Figure 6: Visualising the ancestor recursion. The ancestor function is called from the main program with young=person and old=grandfather2. (1) After checking that young.name != old.name, the ancestor function checks if old is an ancestor of the mother of person by calling the ancestor function with young=mother of person and old=grandfather2. (2) After checking that young.name != old.name, the ancestor function checks if old is an ancestor of the mother of the mother of person by calling the ancestor function with young=grandmother1 and old=grandfather2. The numbers illustrate the sequence of young.name == old.name checks. 4, 5, 7, 9, 10, 13, 14 illustrate that the ancestor is NULL.

**Task 7**. The next command is 'k' to remove all persons born before a given year. Define the function `void removeolderpersons (struct person **families, int birthyear)`
First, you have to remove all mother/father pointers to all persons with a birth date less than the given date. (You can do this by iterating over the linked list and setting the mother or father pointer to NULL when the mother or father is too old.) Next, you have to remove (and free) the persons from the linked list. This is a standard linked-list removal that you're familiar with.
Starting with the family tree of Figure 4, this is an example output:

```
Command? t
name=mother birthyear=1980 mother=unknown father=unknown
name=father birthyear=1975 mother=unknown father=unknown
name=person birthyear=2000 mother=mother father=father
name=baby birthyear=2021 mother=person father=cousin
name=cousin birthyear=2002 mother=unknown father=unknown
Command? k
Cut-off year? 1980
Command? t
name=mother birthyear=1980 mother=unknown father=unknown
name=person birthyear=2000 mother=mother father=unknown
name=baby birthyear=2021 mother=person father=cousin
name=cousin birthyear=2002 mother=unknown father=unknown
Command? k
Cut-off year? 2001
Command? t
name=baby birthyear=2021 mother=unknown father=cousin
name=cousin birthyear=2002 mother=unknown father=unknown
```

**Task 8**. The final task is to print all common ancestors (if any) of two persons. Do not include the two persons themselves. As an example, in Figure 2, only greatgrandfather1 is a common ancestor of aunt2 and uncle3, while the common ancestors of baby and child are: uncle2, greatgrandfather1, grandmother2, grandmother1, and grandfather1. Write a function commonancestors that is called with the 'c' command.

```
int commonancestors (struct person ancestors[MAXANCESTORS],
                     struct person *families, struct person *young1, struct person *young2)
```

This function has a more complicated interface because it must be able to return up to MAXANCESTORS (equal to 128) ancestors. For this reason it has an array input that also serves as output. (In fact, the function only uses the array as output and can just overwrite whatever is in it.) The return value of the function is the number of ancestors in the array.

To implement the function you can iterate over the linked list and check if the current person is an ancestor of both young1 and young2. If it is then you can copy it to the ancestors array.

Here are some example outputs:

```
Command? c
Person1? person
Person2? father
person and father have common ancestors grandmother2, grandfather2.
... and more examples ...
person and person have no common ancestors.
father and mother have no common ancestors.
dodo and baby have no common ancestors.
baby and child have common ancestors uncle2, greatgrandfather1, grandmother2,
grandmother1, grandfather1.
uncle1 and uncle2 have common ancestors greatgrandfather1, grandmother1, grandfather1.
```

**Submission**: Your final solution must be submitted through OnCourse which will automatically grade this submission. Upload your C program to Oncourse (`Exercise 7: Graphs (due 21 Oct 23:55)`). You can resubmit as often as you want until the deadline.

- **12/10 v1.3** Many minor changes for consistency with other homework.
- **14/10 v1.4** Minor fixes.
- **14/10 v1.5** Fixed URL.