tuple.h

```h
#ifndef TUPLE_H
#define TUPLE_H
#include <stdbool.h>

typedef void *poly;
typedef int (*tyEqFn) (poly, poly);
typedef void (*tyOutputFn) (poly);
typedef struct _tuple *tuple;//not same!
tuple newTuple (poly x, poly y);
poly first (tuple t);
poly second (tuple t);
int tupleEquals (tuple t1, tuple t2, tyEqFn eqx, tyEqFn eqy); //equality
testing
void tupleOutput (tuple t, tyOutputFn outx, tyOutputFn outy);
void freeTuple(tuple t) ;

#endif
```

tuple.c

```c
#include <stdio.h>
// This is the standard input-output header. Functions for file input and
output, as well as for standard input and output (like printf, scanf, fopen,
etc.), are declared in this header.
#include <stdlib.h>
#include "tuple.h"

struct _tuple{
    poly firstElement;
    poly secondElement;

};

tuple newTuple( poly x, poly y){
    tuple t = malloc(sizeof(*t));
    if (t == NULL) {
        // Handle memory allocation failure if necessary
        return NULL;
    }

    t->firstElement = x;
```

```
    t->secondElement = y;
    return t;
}

poly first(tuple t){
    return t->firstElement;
}

poly second(tuple t){
    return t->secondElement;
}

int tupleEquals(tuple t1, tuple t2, tyEqFn eqx, tyEqFn eqy){
    return         eqx(t1->firstElement,         t2->firstElement)         &&
eqy(t1->secondElement,t2->secondElement);
}

void tupleOutput(tuple t, tyOutputFn outx, tyOutputFn outy){
    outx(t->firstElement);
    outy(t->secondElement);

}

void freeTuple(tuple t) {
    free(t);
}
```

text.c

```
#include <stdio.h>
#include "tuple.h"

int intEquals(poly x, poly y){
    return *(int*)x == *(int*)y;
}

int charEquals(poly x, poly y){
    return *(char*)x == *(char*)y;
}

int floatEquals(poly x, poly y){
    return *(float*)x == *(float*)y;
}
```

```c
void intOutPut(poly x){
    printf("%d", *(int*)x);
}

void charOutPut(poly x){
    printf("%c", *(char*)x);
}

void floatOutPut(poly x){
    printf("%f", *(float*)x);
}

int main(){
    int x = 5;
    char y = 'a';
    float z = 0.6777;

    tuple t1 = newTuple(&x, &y);
    tuple t2 = newTuple(&y, &z);

    if(tupleEquals(t1, t2, intEquals  , charEquals)){
       printf("Tuples are equal");
    }else {
       printf("tuples are different");
    }

    printf("\n");

    printf("t1:");
    tupleOutput(t1, intOutPut, charOutPut);
    printf("\n");

    printf("t2:");
    tupleOutput(t2, charOutPut, floatOutPut);
    printf("\n");

    freeTuple(t1);
    freeTuple(t2);
    return 0;
}
```

Resule:

```
(base) NLiangs-MacBook-Pro:Tuple a25076$ gcc -o text text.c tuple.c
(base) NLiangs-MacBook-Pro:Tuple a25076$ ./text
tuples are different
t1:5a
t2:a0.677700
```

==========2==========

nTuple.h

```h
#include <stdio.h>
#include <stdlib.h>
#ifndef NTUPLE_H
#define NTUPLE_H


typedef enum {
    INT, CHAR, FLOAT, //... add more as needed
} DataType;

typedef struct {
    void *data;
    DataType type;
} Element;

typedef struct {
    Element *elements;
    int size;
} nTuple;

nTuple* createNTuple(int size) ;

void setElement(nTuple *t, int index, void *data, DataType type) ;

void* getElement(nTuple *t, int index);
DataType getType(nTuple *t, int index) ;

void freeNTuple(nTuple *t) ;

#endif
```

nTuple.c

```c
#include <stdio.h>
#include <stdlib.h>
#include "nTuple.h"

nTuple* createNTuple(int size) {
    nTuple* t = (nTuple *)malloc(sizeof(nTuple));
    t->elements = (Element *)malloc(size * sizeof(Element));
    //t->size = size;
    if (!t) return NULL;

    if (!t->elements) {
        free(t);
        return NULL;
    }

    for(int i = 0; i < size; i++) {
        t->elements[i].data = NULL;
        t->elements[i].type = INT; // default value
    }
    t->size = size;


    return t;
}

void setElement(nTuple *t, int index, void *data, DataType type) {

    if (index < 0 || index >= t->size) {
        fprintf(stderr, "Index out of bounds.\n");
        return;
    }

    t->elements[index].data = data;
    t->elements[index].type = type;
}

void* getElement(nTuple *t, int index) {

    if (index < 0 || index >= t->size) {
        fprintf(stderr, "Index out of bounds.\n");
        return NULL;
    }

    return t->elements[index].data;
```

```
}

DataType getType(nTuple *t, int index) {

    if (index < 0 || index >= t->size) {
        fprintf(stderr, "Index out of bounds.\n");
        return INT; // default value
    }

    return t->elements[index].type;
}

void freeNTuple(nTuple *t) {
    if (t) {
        if (t->elements) {
            free(t->elements);
        }
        free(t);
    }
}

int main() {
    nTuple *t = createNTuple(3);

    int x = 5;
    char y = 'a';
    float z = 0.99;

    setElement(t, 0, &x, INT);
    setElement(t, 1, &y, CHAR);
    setElement(t, 2, &z, FLOAT);

    // Add checks based on getType to correctly process/print the data
    printf("%d\n", *(int*)getElement(t, 0));
    printf("%c\n", *(char*)getElement(t, 1));
    printf("%f\n", *(float*)getElement(t, 2));

    freeNTuple(t);
    return 0;
}
```

Resule:

```
(base) NLiangs-MacBook-Pro:Tuple a25076$ gcc -o nTuple nTuple.c
(base) NLiangs-MacBook-Pro:Tuple a25076$ ./nTuple
5
a
0.990000
```