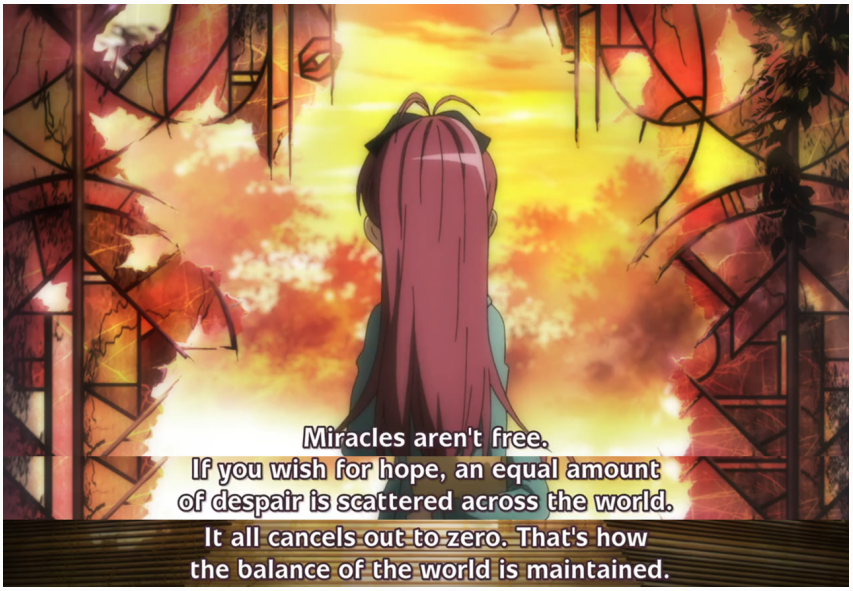
Pandora's Box

> This problem is for CS205 Assignment 2.
>
> Content: **Structure; Pointer; Memory Management**.
>
> Site Fan, 2023.10.24

Background



「奇迹ってのはタダじゃないんだ。希望を祈ればそれと同じ分だけの绝望が撒き散らされる。」
「そうやって差し引きを0にして 世の中のバランスは成り立ってるんだよ。」

Description

> This question is designed to examine your mastery of pointers using C for memory management. You can also use this question as an initial exploration of exception handling for your final project :)

Still frustrated that you can only put the same type of data in an STL vector? Missing Python's lists that can store any type of data? Now, try implementing a Pandora's Box that can store anything of any type in C!

Ofcourse, GuTao is not so boring that letting everyone who is new to C write a complete Pandora's Box from scratch, that would be too difficult. So GuTao decided to give you the blueprint of the Pandora's Box, your task is just to implementseveral key operations to manage and interact with the `PandoraBox` structure.

The `PandoraBox` should provide the flexibility to append, write, and read data of different types, all stored within the same container.

In consideration of robustness, data written to the `PandoraBox` should be correctly handled even when its size doesn't match the item size, including handling data padding and negative values.

The following operations should be supported by your `PandoraBox`:

1. `create`: Create an empty `PandoraBox` with specified item counts and their respective sizes.
2. `append`: Extend the memory of the `PandoraBox` to append an item.
3. `write`: Write data into a specific item within the `PandoraBox`.
4. `read`: Copy the content of a particular item from the `PandoraBox` into newly allocated memory.
5. `destroy`: Properly deallocate the memory associated with the `PandoraBox` and its data to prevent memory leaks.
6. `printc`: Print bytes as characters in a given range, with proper handling of negative values.
7. `printx`: Print bytes as a single hexadecimal value in a given range.
8. `hex2byte`: Convert the given string to hexadecimal, and save it into the given address.

Of course, GuTao is not so boring that letting everyone who is new to C write a complete Pandora's Box from scratch, that would be too difficult. So GuTao decided to give you the blueprint of the Pandora's Box, and you only need to implement a few simple functions.

Input

~~Actually you don't need to worry about this part because the main function is given by GuTao.~~

The first line contains an integer T, the number of operations.

Then the following T lines are operations.

- `C n a[0] a[1] ... a[n-1]`: create a Panbox with n items, the sizes of items are $a[0]\dots a[n-1]$, split by spaces(All indices in this problem start from 0).
- `A l s`: append an item to the Panbox, where l represents the length of a subsequent string, and s represents the hexadecimal string, starting with "0x" and [a-f] in lowercase. For example, `A 9 0x514cafe`.
- `W i l s`: write the item with index i with the value represented in the following hexadecimal string s of length l.
- `R i o`: read the content of the item with index i. If o is 1, then print the item in format of characters, otherwise in format of a single hexadecimal.
- `D`: Open the Pandora's box and free everything, including HOPE.
- `Q`: Display the information of the current Panbox.

Output

~~Actually you don't need to worry about this part because the main function is given by GuTao.~~

- For `R` instructions, print the content of item[i] in the format given by o.
- For `Q` instructions, if the Panbox pointer is NULL, output NULL in a single line; otherwise the number of the items in the Panbox in a single line, then print the size of every item in a new line.
- For `D` instructions, print every item in character format, and in reverse order(FILO), then free the memory.

Sample Input #1

```
12
C 1 4
W 0 10 0x45504f48
A 18 0x6e6f697469626d41
A 24 0x6e6f6974697465706d6f43
A 12 0x6465657247
A 10 0x79766e45
A 18 0x7973756f6c61654a
A 14 0x646572746148
A 20 0x6563697473756a6e49
A 20 0x797265686361657254
A 22 0x68746c6165682d6c6c49
D
```

Sample Output #1

```
I l l - h e a l t h
T r e a c h e r y
I n j u s t i c e
H a t r e d
J e a l o u s y
E n v y
G r e e d
C o m p e t i t i o n
A m b i t i o n
H O P E
```

Hint

To make this problem more interesting, `memcpy` and `memset` of `<string.h>` are banned.

Try to implement them by your self!

```
void *mycpy(void *dst, const void *src, size_t n);
void *myset(void *dst, int c, size_t n);
```

Please **READ THE COMMENT** carefully before implementing.

Template

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

typedef struct PandoraBox
{
    int item_count; // Number of items stored in the PandoraBox.
    int *item_size; // Array of item sizes in bytes.
    void *data;     // A type-agnostic pointer to store data.
} Panbox;

/*
A PandoraBox capable of storing various data types within a continuous memory space.

- item_count: Number of items stored in this box.
- item_size: An array of integers; item_size[i] represents the size (in bytes) of the i-th item in the box.
- data: A type-agnostic pointer, enabling interpretation of different data types by reading segments with varying lengths.
*/

/*=+=+=+=+=+=+=+=+=+=+=+=+=+=Your code starts from here=+=+=+=+=+=+=+=+=+=+=+=*/

void *mycpy(void *dst, const void *src, size_t size)
{
    return NULL;
}
/*
Function: See memcpy in <string.h>

Notes:
- It is not a must to implement this function, but with it you can implement the following functions more easily.
*/

void *myset(void *dst, int n, size_t size)
{
    return NULL;
}
/*
Function: See memset in <string.h>

Notes:
- It is not a must to implement this function, but with it you can implement the following functions more easily.
*/

Panbox *create(int item_count, int item_size[])
```

```
{
    return NULL;
}
/*
Function: Create an empty PandoraBox with specified item counts and their respective sizes.

Parameters:
- item_count: The number of items to be stored in the PandoraBox.
- item_size: An array specifying the size (in bytes) of each item.

Returns:
A pointer to the created PandoraBox.

Note:
- You CAN create a Panbox with no initial items, i.e., item_count=0 and item_size=NULL.
- However, negative item_count is not supported, and item_size should not be NULL when item_count
is non-zero.
- The size of any item should be positive.
- Make sure that the memory newly allocated is filled with 0s.
- You should return NULL immediately if any failure happens.
- Good news: GuTao guarantees that [item_size, item_size+item_count-1] is allocated in advance,
unless item_size is NULL.
*/

void append(Panbox *panbox, void *value, int width)
{
    return;
}
/*
Function: Extend the memory of the PandoraBox to append an item.

Parameters:
- panbox: A pointer to the PandoraBox.
- value: A pointer to the data to be added to the PandoraBox.
- width: The size of the data in value (in bytes).

Note:
- You should not append this item if any failure happens.
- Good news: GuTao guarantees that [value, value+width-1] is allocated in advance, unless value
is NULL.
*/

void write(Panbox *panbox, int item_id, void *value, int width)
{
    return;
}
/*
Function: COPY the data stored in 'value' into the segment of the item with index item_id in the
PandoraBox.

Parameters:
- panbox: A pointer to the PandoraBox.
- item_id: The index of the item in the PandoraBox where data will be written.
```

```
- value: A pointer to the data that needs to be written to the PandoraBox.
- width: The size of the data in value (in bytes).

Notes:
- If 'item_id' is out of valid bounds, the data won't be written.
- If the item size cannot hold all data in value, the data won't be written.
- If the width of value is smaller than the item size, the data will be placed in the lower
address and padded with the highest bit.
- Example: inserting a 1-byte value into a 2-byte item: (0x7f -> 0x007f), (0xf7 -> 0xfff7).
- The item_size should not change in this function.
- You should not write the item if any failure happens.
- Good news: GuTao guarantees that [value, value+width-1] is allocated in advance, unless value
is NULL.
*/

void *read(Panbox *panbox, int item_id)
{
    return;
}
/*
Function: COPY the item with index item_id from the PandoraBox into a newly allocated memory, and
return a pointer to this memory.

Parameters:
- panbox: A pointer to the PandoraBox.
- item_id: The index of the item in the PandoraBox that should be read.

Returns:
A pointer to the newly allocated memory containing a copy of the specified item.

Notes:
- You should return NULL immediately if any failure happens.
*/

void destroy(Panbox *panbox)
{
    return;
}
/*
Function: Deallocate the memory associated with the PandoraBox and its data.

Parameters:
- panbox: A pointer to the PandoraBox that should be destroyed.

Notes:
- This function should be used when the PandoraBox is no longer needed to prevent memory leak.
- You should check before freeing a pointer.
*/

void printc(void *value, int width)
{
    return;
}
```

```
/*
Function: Print every 1 byte (char) in [value, value+width-1] as a character, split by a space.

Notes:
- Good news: GuTao guarantees that [value, value+width-1] is allocated in advance, unless value
is NULL.
- If the value of the byte is in [0, 32] or equals to 127(Del), do not print anything(they are
special characters).
- e.g., for an empty box with all 0, the destroy of this box will invoke printc(), but should not
print anything.
- If the value of the byte is negative, print '-' instead.
- You should return immediately if any failure happens.
- Please print a `\n` before returning, unless you print nothing in this function.
*/

void printx(void *value, int width)
{
    return;
}
/*
Function: Print the value in [value, value+width-1] as a single hexadecimal, in format of
0x12345678.

Notes:
- Good news: GuTao guarantees that [value, value+width-1] is allocated in advance, unless value
is NULL.
- You should return immediately if any failure happens.
- If the highest byte is less than 0x10, please output a leading 0, e.g., 0x0514.
- Leading zeros are OK, if *(value) = 0x000f, width = 2, then print 0x000f. (print even number of
characters)
- If the item is full of 0s, please still print it. E.g., 0x0000.
- Please print a `\n` before returning.
*/

void hex2byte(void *dst, char *hex)
{
    return;
}
/*
Function: For the given hex string "0x12345678", write 0x12345678 into dst.

Notes:
- The length of hex might be ODD, make sure to put all data into dst. For example, "0xfff"(12
bits) should be stored using 2 bytes as 0x0fff(different from write(), just add a padding 0
here).
- You should return immediately if any failure happens.
*/

/*=+=+=+=+=+=+=+=+=+=+=+=Your code ends here=+=+=+=+=+=+=+=+=+=+=+=*/

void show_info(Panbox *p)
{
    if (p != NULL)
```

```
    {
        printf("%d\n", p->item_count);
        for (int i = 0; i < p->item_count; i++)
        {
            printf("%d ", p->item_size[i]);
        }
        printf("\n");
    }
    else
    {
        printf("NULL\n");
    }
}
/*
Function: Display the basic information of given PandoraBox

Implemented by GuTao.
*/

Panbox *p = NULL;

int main()
{
    int T;
    scanf("%d", &T);
    while (T--)
    {
        char op;
        scanf(" %c", &op);
        switch (op)
        {
        case 'C': {
            int item_count;
            scanf("%d", &item_count);
            int *item_size = (int *)malloc(item_count * sizeof(int));
            for (int i = 0; i < item_count; i++)
            {
                scanf("%d", &item_size[i]);
            }
            p = create(item_count, item_size);
            free(item_size);
            break;
        }
        case 'D': {
            if (p != NULL)
            {
                for (int i = p->item_count - 1; i >= 0; i--)
                {
                    void *data = read(p, i);
                    if (data != NULL)
                    {
                        printc(data, p->item_size[i]);
                        free(data);
```

```
                }
            }
        }
        destroy(p);
        break;
    }
    case 'A': {
        int len;
        scanf("%d", &len);
        char *hex_str = (char *)malloc(len + 1);
        scanf("%s", hex_str);
        int num_of_byte = (strlen(hex_str) - 1) / 2;
        void *data = (void *)malloc(num_of_byte);
        hex2byte(data, hex_str);
        append(p, data, num_of_byte);
        free(hex_str);
        free(data);
        break;
    }
    case 'W': {
        int item_id;
        scanf("%d", &item_id);
        int len;
        scanf("%d", &len);
        char *hex_str = (char *)malloc(len + 1);
        scanf("%s", hex_str);
        int num_of_byte = (strlen(hex_str) - 1) / 2;
        void *data = (void *)malloc(num_of_byte);
        hex2byte(data, hex_str);
        write(p, item_id, data, num_of_byte);
        free(hex_str);
        free(data);
        break;
    }
    case 'R': {
        int item_id;
        scanf("%d", &item_id);
        int is_printc;
        scanf("%d", &is_printc);
        void *data = read(p, item_id);
        if (data != NULL)
        {
            if (is_printc)
            {
                printc(data, p->item_size[item_id]);
            }
            else
            {
                printx(data, p->item_size[item_id]);
            }
            free(data);
        }
        break;
```

```
        }
        case 'Q': {
            show_info(p);
            break;
        }
        default: {
            break;
        }
        }
    }
    return 0;
}

/*
Main function implemented by GuTao, you cannot rewrite one for yourself.

- Just for you to understand how the functions you implement would be invoked.
- Not exact the main function to test on OJ, this one does not check your error handling.
- You can test your functions on your PC with your own main().
- You may want to hack with your own main function, but this won't help you AC :)
*/
```

Test cases

1. NULL pointer and invalid arguments handling for each function, create and destroy empty box.
2. Create, write items using data of exact lengths, read items, printc, destroy
3. Create, write items using data of exact lengths, read items, printc/x, destroy
4. Create, write items using data of random lengths, read items, printc, destroy, negative char
5. Create, write items using data of random lengths, read items, printc/x, destroy, negative char
6. Create, write items with invalid indices, append items, read items with invalid indices, printc, destroy
7. Create, write items with invalid indices, append items, read items with invalid indices, printc/x, destroy
8. Create, append items, write items, read items, printc/x, destroy, with full robustness
9. Memory leak check: (create, append items, destroy) *n
10. Large amount of create, append items, write items, read items, printc/x, destroy, with full robustness