VG101 Lab Worksheet

Lab 7

Instructor: Dr. Yifei ZHU
TA: Hangrui CAO
TA: Haoxuan SHAN
TA: Qinhang WU
TA: Muchen XU

Code Quality Requirement

- Ensure proper indentation
- Make naming meaningful
- Include necessary comments
- Split the code over functions
- Test the code as much as you can

Basic Syntax Exercises

Malloc Warm Up

Use `malloc` to create an array to storage `n` numbers. The input is a number `n` and followed by `n` decimal numbers (Use double).

Point structure

1. Construct a `Point` structure, storing the x,y as coordinates for this point.
2. Design a function `Point CreatPoint(double x,double y)`. Return a new `Point` with coordinate x,y.
3. Design a function `double PointDistance(Point p1,Point p2)`. Return the distance between two points.
4. Design a function that can exchange the value of `x` and `y` of a Point with the usage of pointers.

Rectangle structure

1. Include two Points (left-down and up-right).
2. Design a function `CreateRectangle(Point p1, Point p2)`. (p1, p2 may be the other pair of diagonal points)
3. Design a function return the size of the Rectangle.
4. Design a function return the size of the overlap area of two rectangles.

Practical Exercise

Adding (JOJ)

Given two integers x and y, print the sum of them.

Input: Two integers x and y, satisfying $0 <= x, y <= 32767$.

Output: One integer, the sum of x and y.

Sample Input:

```
3 5
```

Sample Output:

```
8
```

Simple Vector

Here we will realize an integer array called `vector`, whose length (size) varies dynamically, according to the usage.

The basic strategy follows that

1. Record `size` as the number of positions used in the array, `limit` as the real capacity. A memory of `limit` blocks of integers is always allocated and assigned the first chunk to the integer pointer `a`.
2. Initialize an empty vector as `size=0` and `limit=1`.
3. If we want to use the position beyond `limit`, we should double the capacity `limit`. (Ask for allocating and move the memory form the origin place to the new place.)
4. If the current `size` is no more than half of `limit`, we will half the `limit`. (Reallocate the memory)
5. Always remember free the unused memory.

And we want our `vector` to be able to

- `push_back`: push a new element to the back of the vector
- `pop_back`: remove the last element of the vector
- `get`: get the element in the vector by index
- `change`: change the element in the vector by index
- `print`: print all the element in the vector in one line

Starter file:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct vector_t
{
    int *a; // first place of the array
    unsigned int size, limit;
} vector;
vector construct()
{
    //TODO: construct an empty vector
}
void destruct(vector *v)
{
    //TODO: destruct a vector (free the allocated memroy)
```

```
}
void extend(vector *v)
{
    //TODO: make the vector as twice long as it is now
}
void contract(vector *v)
{
    //TODO: make the vector as half long as it is now
}

void push_back(vector *v, int element)
{
    //TODO: push a new element to the back of the vector
    //      remember to extend the vector if its size reaches the limit
}
void pop_back(vector *v)
{
    //TODO: remove the last element
    //      remember to contract the vector if its
    //      size reaches half of the limit
}
int get(vector *v, unsigned int index)
{
    //TODO: get the element of the vector at index
    //      if the index exceeds the size, return 0;
}
void change(vector *v, unsigned int index, int element)
{
    //TODO: change the term of the vector at index to element.
    //      if the index exceeds the size, do nothing;
}
void print(vector *v)
{
    //TODO: print all the elements in the vector in a line
    //      and then make a new line.
}
int main()
{
    // Simple test
    vector w = construct();
    for (int i = 0; i < 10; i++)
    {
        push_back(&w, i);
        print(&w);
    }
    for (int i = 0; i < 5; i++)
    {
        change(&w, i, i + 5);
        pop_back(&w);
        print(&w);
    }
    destruct(&w);
    return 0;
}
```

Simple Set (optional)

Here we will realize an integer set, which just behaves like a mathematical set.

We want our `set` to be able to

- `insert`: insert a new element to the set
- `remove`: remove a specific element in the set
- `in`: check whether an element belongs to the set
- `print`: print all the elements in the set in one line

Starter file:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
typedef struct set_t
{
    //TODO: Design yourself
} set;
set construct()
{
    //TODO: construct an empty set
}
void destruct()
{
    //TODO: destruct a set
}
int in(set *s, int element)
{
    //TODO: return 1 if set s has this element
    //      otherwise, return 0
}
void insert(set *s, int element)
{
    //TODO: insert the element to the set s
}
void erase(set *s, int element)
{
    //TODO: remove the element in the set s
    //      if s doesn't have this element, do nothing
}
void print(set *s)
{
    //TODO: print all the elements in the set in a line (in any order)
    //      and then make a new line.
}
int main()
{
    // Simple test
    set w = construct();
    for (int i = 0; i < 10; i++)
    {
        insert(&w, i);
        print(&w);
    }
    for (int i = 0; i < 10; i += 2)
    {
        erase(&w, i);
```

```
47          print(&w);
48      }
49      destruct(&w);
50  }
```

Or you may modify the code you previously complete for the vector.