COMP10002 Foundations of Algorithms

Workshop Week 9

Wenbin Cao

September 26, 2019

GitHub Repo: https://github.com/AlanChaw/COMP10002-FoA

Outline

- Discussion Questions for Structs
- Review on Dynamic Memory
- Programming Exercises for Dynamic Memory

Discussion

People have titles, a given name, a middle name, and a family name, all of up to 50 characters each. People also have dates of birth (dd/mm/yyyy), dates of marriage and divorce (as many as 10 of each), and dates of death (with a flag to indicate whether or not they are dead yet). Each date of marriage is accompanied by the name of a person. Assuming that people work for less than 100 years each, people also have, for each year they worked, a year (yyyy), a net income and a tax liability (both rounded to whole dollars), and a date when that tax liability was paid.

Countries are collections of people. Australia is expected to contain as many as 30,000,000 people; New Zealand as many as 6,000,000 people.

Discussion

Exercise 1

Give declarations that reflect the data scenario that is described.

Discussion

Exercise 2

Write a function that calculates, for a specified country indicated by a pointer argument (argument 1) with a number of persons in it (argument 2), the average age of death. Do not include people that are not yet dead.

Discussion

Exercise 3

Write a function that calculates, for the country indicated by a pointer argument (argument 1) with a number of persons in it (argument 2) the total taxation revenue in a specified year (argument 3).

Now that you see the processing mode implied by this exercise, do you want to go back now and revise your answer to Exercise 1? If you did, would you need to alter your function for Exercise 2 at all?

Dynamic Memory

- We always cannot know how much memory we need in advance.

- The functions:

```
size_t sizeof(thing)  // can get either a type or a variable

void *malloc(size_t size)

void *realloc(void *ptr, size_t size)

void free(void *ptr)
```

```
double A[10]; char *p="mary mary quite contrary";
printf("sizeof(char)   = %2lu\n", sizeof(char));
printf("sizeof(int)    = %2lu\n", sizeof(int));
printf("sizeof(float)  = %2lu\n", sizeof(float));
printf("sizeof(double) = %2lu\n", sizeof(double));
printf("sizeof(A)      = %2lu\n", sizeof(A));
printf("sizeof(*A)     = %2lu\n", sizeof(*A));
printf("sizeof(p)      = %2lu\n", sizeof(p));
printf("sizeof(*p)     = %2lu\n", sizeof(*p));
```

malloc

```
void *malloc(size_t size)
```

Example, initialize an array dynamically

```
int n;
printf("input the size for the integer array: ");
scanf("%d", &n);

int *p = (int *)malloc(n * sizeof(int));

/* do something  */
/* do something  */
/* do something  */

free(p);
p = NULL;
```

realloc

```
void *realloc(void *ptr, size_t size)
```

- Allocates a fresh segment of memory contains *size* bytes
- Copies the memory segment indicated by *ptr* to this new segment
- Frees the segment indicated by *ptr*

realloc

Example, realloc sapce

```
int original_size = 10;
int new_size = 100;

char* p = (char *)malloc(original_size);
printf("%p\n", p);

p = (char *)realloc(p, new_size);
printf("%p\n", p);
```

Output:

```
0x10330b1f0
0x1033092c0
```

Exercises

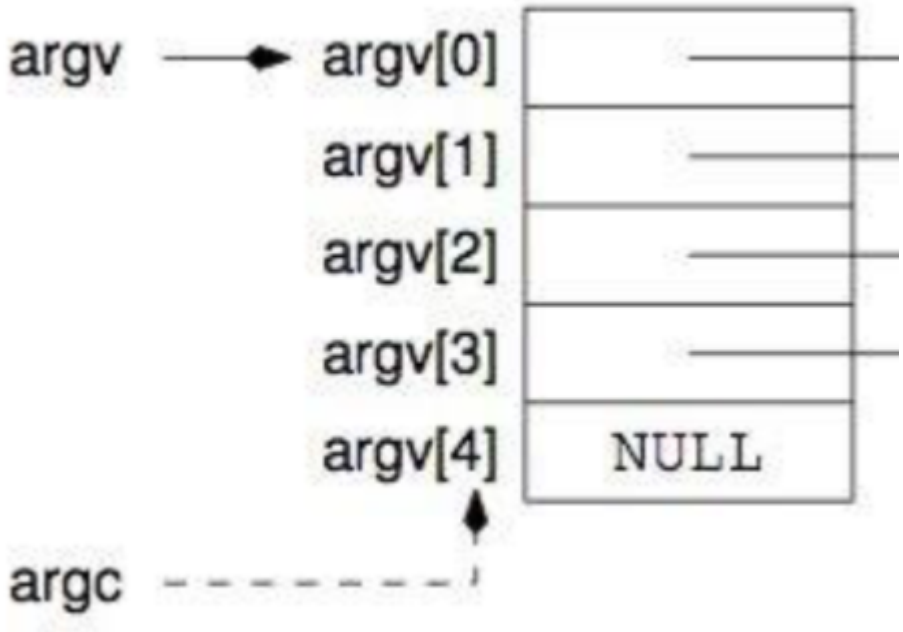
Exercise 4

Write a function `char *string_dupe(char *s)` that creates a copy of the string `s` and returns a pointer to it.

Exercise 5

Write a function `char **string_set_dupe(char **S)` that creates a copy of the set of string pointers `S`, assumed to have the structure of the set of strings in `argv` (including a sentinel pointer of `NULL`), and returns a pointer to the copy.

The structure for "argc" and "argv"



Exercises

Exercise 6

Write a function `void string_set_free(char **S)` that returns all of the memory associated with the duplicated string set `S`.

Exercise 7

Test all three of your functions by writing scaffolding that duplicates the argument `argv`, then prints the duplicate out, then frees the space.

(What happens if you call `string_set_free(argv)`? Why?)