

Tutorial 6

Memory management

Jiawei Li (Michael)

Office hours: Tuesday 3:00 – 5:00pm

Office: PMB426

Email: jiawei.li@nottingham.edu.cn

Lab 5 word guessing game

When the program starts, it will prompt to read in an integer which is the number of words in the word list. The program then reads in that number of words. You can assume that all the words in the game have a maximum of 31 characters. After they have all been entered, the program randomly selects a word to use for the next game.

Buffer

- *A **temporary storage area** is called buffer. All standard input and output devices contain an input and output buffer. In standard C/C++, streams are buffered, for example in the case of standard input, when we press the key on keyboard, it isn't sent to your program, rather it is buffered by operating system till the time is allotted to that program.*

For the example program we run, what are stored in the buffer?

'a' '\n' 'b' '\n' ...

- *'a'* -> &arr[0];
- *'\n'* -> &arr[1];
- *'b'* -> &arr[2];

How to solve it?

Process structure in memory

Stack

Data area that grows downwards towards the heap

LIFO data structure, for local variables and parameters

Heap

Data area that grows upwards towards stack

Specially allocated memory (malloc, free, ..., probably new, delete)

Data and BSS (uninitialised data) segment

Read-only:	Constants	String literals
Read/write:	Global variables	Static local variables

Code (or text) segment

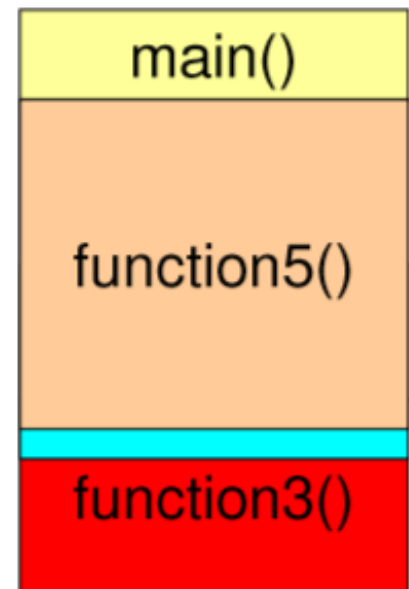
The program code

Stack and heap

- Local variables and function parameters are allocated on the **stack**. This is automatically done by the C compiler.
- The stack is fully controlled by the operating system.
- Dynamic memory allocation on the heap needs to be done manually.

The stack

- A stack is a last-in-first-out (LIFO) structure
- Like a stack of books
 - You add to the top
 - You take from the top
- Function calls (stack frames) are stored on a stack in memory
- Aside: Note that **most** stacks go down in memory addresses
 - i.e. the stack frame for the new function is lower in memory



Memory allocation

- Allow us to allocate data dynamically
- When and why should we allocate data on the heap?

```
int main()
{
    int a;
    int* ptr = malloc(sizeof(int));
    ...

    free(ptr);
}
```


5 steps to dynamic memory allocation

Step 1. Work out how much memory you need to allocate

- remember the `sizeof()` operator!

Step 2. Ask for the amount of memory

- use `malloc (memory_size)`

Step 3. Store the returned pointer e.g.:

```
int* pInt = malloc( sizeof(int) );
```

Step 4. Use the memory through the pointer, as if it was the correct type

```
*pInt = 5; (*pInt)++; pInt = p;
```

Step 5. when finished, free the memory

```
free( pInt );
```

Frequent mistakes

- **Malloc() and free() should be used in pair. Error occurs when we mistakenly free up allocated chunk or we try to free up a part of it.**

```
{    int *ptr;
    int n = 10;

    ptr = malloc(n * sizeof(int));
    ...

    free(ptr+5);    // want to free part of ptr

    free(ptr);

    ...
    free(ptr);    // free up more than once
}
```

Good use of malloc

- **Verify whether memory allocation is successful or not before dereferencing the pointer.**
- Dynamic allocation function returns **NULL** in the event of unsuccessful memory allocation (insufficient memory available for example).

```
int main()
{
    int *ptr;
    ptr = malloc(100 * sizeof(int));

    if(ptr != NULL) {
        ...
    }
    else {
        ...
    }
}
```

Exercise 1 Figure out what this program does

```
int main()
{
    int i,n;
    float *element;
    scanf("%d", &n);
    element = malloc( n*sizeof(float) );
    if(element==NULL) {
        exit(0);
    }
    for(i=0;i<n;++i)
    {
        scanf("%f",element+i);
    }
    for(i=1;i<n;++i)
    {
        if(*element<*(element+i))
            *element=*(element+i);
    }
    printf(" The wanted element is : %.2f \n\n",*element);
    free(element);
    return 0;
}
```

More debugging exercises

Three programs with the names 'incorrect1.c', 'incorrect3.c', and 'incorrect4.c' are available on the Moodle page. Try to find out bugs in three programs. The first program should output the quotient and remainder of the division of two values. The second program is to compare the record of two students (using C struct). The third program is the Rogue-like game (in Lab 7).