VG101: Introduction to Computer and Programming

Midterm Review

Second Midterm

- 100-minute open-book paper exam
- Question types see sample exam on Canvas
- Easy

Header files

- Common libraries:

```
#include <stdio.h>   // For standard input and output.
#include <math.h>    // For basic mathematic operations like sqrt, sin,
pow.
#include <stdlib.h>  // For random number generation and dynamic memory
allocation.
#include <string.h>  // For efficient string manipulation.
#include <time.h>    // For recording program running time.
```

- Difference between `<>` and `""`

`main` function

- `main` function is where a program starts executing.
- A program only contains **one** `main` function.
- `main` function is not callable for you. (No recursion for `main` function)
- Return type and return value for `main` should be `int` and `0` (non-zero return value is to indicate some errors)

```
int main() // Return type: a integer
{
    // Write your code here
    return 0; // Return value: 0
}
```

Variable

Variable name

- First character must be letter or `_`
- Followed by letter, number, or `_`
- Case sensitive. `a` and `A` are different variable names.

Variable type

- Basic variable types `int`, `long long int`, `double`, `float`, `char`
- Pointer types `int *`, `double *`, `void *`
- Constant `const int`, `const int *`, `int const *`
- Array `int a[10]`
- Structure type
- Question

```
// what is the variable type of a, b?
int * a, b;
```

Variable initialization

- Give value when declaration `int a = 1`, `int * b = NULL`
- Different variables split by `,`
- Wrong: `int a = b = 1` if `b` undeclared before.

Variable scope

- Start from declaration, end with the scope (block)
- Local variable, global variable, static local variable

Operations

Basic operations

- Assignment operator: `=`
- Arithmatic: `+`, `-`, `*`, `/`
- Remainder: `%`
- Increment, decreament: `++`, `--`
- Shorthand assignment operators `a += b`
- `(condition) ? expression1 : expression2`
- Logic

operator	operation
&&	And
||	Or
!	Not
==	Equal to
!=	Not equal to

- dereference, reference: `*` and `&`
- Question

```
// Right or wrong?
int ax, bx, cx;
1 = ax;
bx = cx = 1;
```

Other operations

- Data type conversion
  - Automatically data type conversion
  - Manually data type conversion
  - Question

```
// What's the value of each variable
int a = 1 + 1.5;
double b = 3 / 2;
double c = 3 / 2.0;
int d = a + b + c;
char e = 'a';
e = e + 1;
// What's the value of integerPi?
double pi = 3.1415926;
int integerPi = (int)pi;
```

Control statements

- `if`, `switch`, `while`, `for`
- `continue`, `break`
- Mind the place of `;`
- Block `{}` : enclose multiple statesments
- Block `{}` can be omitted if it only enclose one statement.
- Again, **Indent is important and useful!**

- In midterm2, TA hand grade your code. No indent -> messy -> grading mistakes.

Function

- Syntax:

```
int add(int a, int b)   // returnType functionName (parameterType
parameter, ...)
{                       // block enclose function content
    return (a + b);     // return value (must consistent with returnType)
}
```

- Parameter passed by value
- Same variable name can be used in different function (they do not share value)
- If the inside variable share the same name with the outside variable, it will block the outside one
- Use void function if function return nothing.
- Use void parameter if function needs no input parameters. (can be omitted)

```
void helloWorld(void)           // No input parameter, no return value.
void helloWorld() is also okay
{
    printf("Hello world\n");    // Only do something.
}
```

I/O

- Include `<stdio.h>`
- Output function: `printf`
- Input function: `scanf`
- File output function: `fprintf`
- File input function: `fscanf`
- File pointer `FILE *`

Arrays

- An **ordered** collection of data values of **the same type**.
- Size must be a constant number.

Declaration and initialization

- Declare an array: type, name, size.
- Macro define: `#define MAX_SIZE 100`
- Ways of initialization

```
int zeros[20] = {0};                    // All elements initialized as 0
int notAllZero[20] = {1};               // Only the first one will be 1,
other will be 0
int numbers[5] = {1, 2, 3, 4, 5};
int numbers[] = {1, 2, 3, 4, 5};        // Equivalent to previous
char str[4] = {'a', 'b', 'c', '\0'};    // Define a string, '\0' is ASCII
0
int numbers[5];
for (int i=0; i<5; i++)
    numbers[i] = i+1;
```

Element accessing

- Index starts with 0.
- Can use integer variable/expression to as array index
- Be careful: array out of bounds!

Two-dimensional Array

- An array whose elements are arrays.
- Use `A[i][j]` to access elements.
- Row first, column second.
- Stored as one dimensional array in memory.
- Address issue is more complicated.
- Higher-dimensioanl array has same property.

C-style string

- An character array end with `'\0'` (0 in ASCII)
- Always remember to keep a place for `\0` (it is also a char)
- Difference between char array and C-style string
- Common string function (`<string.h>`)
    - `fgets`
    - `strlen`
    - `stpcpy` , `strcat` , `strcmp`
    - `strchr` , `strstr`
    - `atoi` , `atof` , `atol`

Pointer

Declaration and Assignment

- Pointer declaration `int * px`
- Pointer assignment `px = &x`
- `NULL` pointer
- `void *` pointer

Array with Pointer

- Array name is a pointer!
- `array[3]` is equal to `*(array + 3)`
- A two dimension array is essentially an array of pointer.
- The type of a constant C-string `"Hello World!"` is essentially a `const char*`
- Question

```
// Which element will be printed?
int a[5][5];
printf("%d\n", *(a+10));
```

Advanced pointer structures

- pointer to pointer `int ** pt`
- array of pointers `int* pt[10]`

Dynamic Memory Allocation

- `#include <stdlib.h>`
- `malloc()` and `free()`
- Use `sizeof()` to calculate size.

```
int *arr = (int *) malloc(10*sizeof(int));      // allocate an piece of
memory
                                                // with the size of 10
ints
free(arr);                                      // free the memory
```

- Life cycles of allocated memory: start with `malloc()`, end with `free()`
- You **must** free all the memory you allocate.

Structure

- Assemble a package of variables into a structure.
- Two ways to define strctures (they are equivalent):

```
struct Student
{
    char name[100];
    int studentID[12];
    int midtermScore;
};
struct Student t;
```

```
typedef struct
{
    char name[100];
    int studentID[12];
    int midtermScore;
} Student;
Student t;
```

- Access member data
  - use `.` to visit member data. `t.midtermScore`
  - use `->` if using pointer. `pt->midtermScore`

Tips

- Arrogance instead of ignorance is your enemy. (Especially in an easy exam)
- Don't forget to add `;`
- Do **intend** your code