## COMP2113 Programming Technologies / ENGG1340 Computer Programming II **Module 10.** C programming (Part 3) – Memory allocation and struct

#### **Objectives**

At the end of this self-learning lab, you should be able to:

- Know how to perform dynamic memory allocation in a C program.
- Know how to use struct to model an object in a C program.



#### Section 1. Dynamic memory allocation

- In C++, we perform dynamic memory allocation with the new operator.
- C does not provide the new operator 3.
- Instead, it uses the following malloc() function.

void \* malloc ( int size );

- The input parameter int size of the malloc() function specifies the number of bytes of memory required.
- Note that the function returns void \*, which means that malloc() returns a pointer to the space allocated (the pointer is not defined as particular data type), or returns null if the memory allocation fails.
- We need to include the stdlib library to use malloc(), free(), and NULL.

#include <stdlib.h>

- Let's browse to /module10/
- \$ cd module10
- Consider memory1.c as an illustration.

```
$ vi memory1.c
```

```
#include <stdio.h>
#include <stdlib.h>
int main(){
    int size;
    int *a;
    printf("How many slots in the array?");
    scanf("%d", &size);

2    a = (int*) malloc ( size * sizeof(int) );
    for( int i = 0; i < size; i++)
        a[i] = i;
    for( int i = 0; i < size; i++)
        printf( "%d ",a[i]);
}</pre>
```

#### **Code explanations:**

- Define a as a pointer to an integer (for pointing to the newly allocated memory in the next line).
- Request memory for storing "size" number of integers.
  - o sizeof (int) function returns the number of bytes required by each int.
  - Therefore "size \* sizeof(int)" returns the number of bytes required to store
     "size" number of integers.
  - o malloc() then allocates memory for storing "size" number of integers, and returns a pointer to the newly allocated memory.
  - o Since the pointer is of the type void, we use (int\*) to cast the void pointer to the integer type pointer, so that we can assign it to the integer pointer a.
  - o Please note that this is a standard way to request for memory in C program. The same format applies for requesting memory for double, char, and even user defined struct.

• Let's try to compile the program and run it.

```
$ gcc -std=c99 memory1.c -o memory1
$ ./memory1
How many slots in the array?10
10
0 1 2 3 4 5 6 7 8 9
$ ./memory1
How many slots in the array?20
20
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
```

This is a common technique in C for dynamic memory allocation, useful for implementing dynamic resizing array, and also many data structures.



#### **Release memory**

• We can release the memory obtained through malloc() by free().

# void free( void \*ptr );

- We should ensure that the pointer value passed to free () is the value returned through a malloc() function.
- For example, the following code requested 10 array slots and release the 10 array slots.

```
int * a = (int *) malloc ( 10 * sizeof(int) );
...
free(a); // All the 10 slots of memory are released
```

• Freeing memory not allocated by malloc() will cause a runtime error.

- C does not have classes.
- To group a number of items into a new type, C uses structs.
- There is no access control with structs. All members are public and can be accessed anywhere in the program.
- structs cannot have member functions.
- structs do not have constructor or destructor. Hence, if we want to initialize an object of a struct, we need to define a function for initializing the object.
- In summary, a struct is simply a group of data and is not designed to support object-oriented programming.
- Consider the file struct1.c

\$ vi struct1.c

```
#include <stdio.h>
#include <string.h>
struct student{
    char name[20];
    int uid;
};

int main(){
    struct student a; //create an object of student
    strcpy( a.name, "Kit" );
```

a.uid = 2012111111;

#### **Code explanations:**

- Define a student structure, the syntax is the same as C++, note that we have a semi colon after the definition of the structure.
- Create a student object.
  - o To create an object of the struct, note that we need to repeat the struct keyword before the name student.

printf( "%s has uid %d.\n", a.name, a.uid );

- o This is required by the C syntax. If we do not want to repeat the struct keyword every time we create an object, we can define Student to be a type using typedef.
- Let's try to compile the program and run it.

```
$ gcc struct1.c -o struct1
$ ./struct1
Kit has uid 2012111111.
```

#### The use of typedef

- typedef defines an alias, it is simply shorthand for programmers.
- Consider the following typedef example.

### typedef struct student Student;

- The typedef keyword creates an alias, instructing the compiler to treat "Student" as "struct student".
- Therefore, we can simply use "Student a" to create a student object because "Student a;" equals to "struct student a;". This is a more intuitive way to construct a Student object.
- Consider the following example:

```
$ vi typedef.c
```

```
#include <stdio.h>
#include <string.h>
struct student{
    char name[20];
    int uid;
};

typedef struct student Student;
int main(){
    Student a; //create an object of student
    strcpy( a.name, "Kit" );
    a.uid = 2012111111;
    printf( "%s has uid %d.\n", a.name, a.uid );
}
```

#### **Code explanations:**

- Define an alias: Student = "struct student"
- When the compiler reads Student a, it considers it as struct student a.
- Let's try to compile the program and run it.

```
$ gcc typedef.c -o typedef
$ ./typedef
Kit has uid 2012111111.
```

- Let's create an application that stores an array of Employee structures in C.
- Each Employee object should contain the following information:
  - 1. Name
  - 2. Position
  - 3. Salary



• Let's create a new blank file called employee.c

```
$ vi employee.c
```

• Task 1. Include the necessary libraries. There are three libraries required

```
#include<stdio.h>
#include<???>
#include<???>
```

- o #include<stdio.h> for handling input and output (printf() and scanf()).
- o #include<???> for handling C-string (e.g., strcpy())
- o #include<???> for handling dynamic memory allocation (e.g., malloc(),
   free(), NULL)

#### • Task 2. Define the Employee structure

```
struct Employee{
    // Three lines of codes here...
};
```

- We need to use char array to store the Name and Position of each Employee.
   Let's initialize the array size as 100 slots.
- o Let's use int type variable to store the Salary.
- Please don't miss the semi-colon after the definition of the Employee structure
   .

#### • Task 3. Define an alias for "struct Employee"

```
typedef ???;
```

- We do not want to type "struct Employee" everytime we create an Employee object or pointer to an Employee object.
- o Let's make "Employee" as an alias of "struct Employee". Note that it is also possible to have the alias name the same as the structure name ©.

- Task 4. Define a function for setting the value of the member variables of an Employee object.
  - As C does not support class, we cannot define member function for Employee,
     let's make setEmployee() as a general function.

```
void setEmployee(char n[], char p[] , int sal, ???) {
    // 3 lines of codes here
}
```

- o The first 3 parameters are **passing by value**, n, p and sal contains the values of the Name, Position and Salary of the Employee e.
- o The 4<sup>th</sup> parameter has to **pass by reference**; we are passing the address of an Employee object into this function, so the input parameter should be a pointer to an Employee object.

```
Employee *e
```

o 1st line of code - Copy the value in char array n into the Name of the Employee object e.

```
strcpy(<mark>???</mark> , n);
```

- Hints: Same as C++, since e is a pointer, when accessing the member variable of the object pointed to by the pointer e, we can use the "->" operator.
- o  $2^{nd}$  line of code Copy the value in char array p into the Position of the Employee object e.

```
strcpy(???, p);
```

o  $3^{rd}$  line of code – Assign the value of sal to the Salary of the Employee object e.

```
e->Salary = sal;
```

Task 5. Define a function printing the information of an Employee object.

```
void showInfo(Employee e) {
    // 3 lines of code here ...
}
```

- Note that we are passing the Employee object e by value, because we do not need to update the input object e. We just need to access its member variables and print it on screen.
- o  $1^{st}$  line of code print the Name of the Employee e.

```
printf("Name: %s\n", e.Name);
```

- Note that as e is not a pointer in the showInfo() function, we can simply access the member variables of e by using member access operator "." (a dot).
- e. Name is a char array, so we use the %s conversion specifier.
- o  $2^{nd}$  line of code print the Position of the Employee e.
- $\circ$  3<sup>rd</sup> line of code print the Salary of the Employee e. Note that e. Salary is an integer value. Please use the right conversion specifier to display the integer value in the string literal.

• Task 6. In the main() function, read in user input.

```
int main() {
    // More code here...
}
```

o The first input is an integer that tells the number of employee in the company.

```
int numOfEmployee;
scanf( ??? , ???);
```

- Note that we need to provide the conversion specifier to indicate that we are reading an integer value.
- Remember to pass in the "address of" variable numOfEmployee to the scanf() function. (We pass numOfEmployee by reference so that scanf() can update the value of numOfEmployee).
- Task 7. Create an array of Employee with numOfEmployee number of slots.

```
Employee *e;
e = (???) malloc ( ??? * sizeof( ??? ));
```

Task 8. Use a for loop to read in all Employee information from user input.

```
char n[100], p[100];
int sal;
for (int i = 0 ; i < numOfEmployee ; i++) {
    scanf("%s%s%d",n,p,&sal);
    setEmployee( ??? , ??? , ???);
}</pre>
```

- o scanf("%s%s%d",n,p,&sal);
  - Reminded that since n and p are array of char, we do not need the "address-of" operator to retrieve their address, because they are already pointer storing the address to the first slot of the char array.
  - On the other hand, sal is a normal int type variable, we need to use &sal to pass in the address of sal to the scanf() function.
- o What are the four parameters in the setEmployee () function?
  - Note that the 4<sup>th</sup> parameter of setEmployee() is requiring an address value (because it is pass by reference). We need to provide the address of the i-th slot of the Employee array e.
  - Therefore we can use either one of the method below, we can pass the address of the i-th slot of the array.

```
setEmployee( n , p , sal , &e[i]);
```

Or we can simply pass in the value of the pointer pointing to e+i.

```
setEmployee( n , p , sal , e+i);
```

• Task 9. Use a for loop to print the information of each Employee.

```
for (int i = 0 ; i <numOfEmployee ; i++) {
    showInfo( ??? );
}</pre>
```

o Note that we are simply pass an Employee object by value, so we can simply pass in the i-th slot of the Employee array e to the showInfo() function.

```
showInfo( e[i] );
```

• Task 10. Finally, free the memory we requested.

```
free(e);
```

- $\circ$  This will release the memory of the entire array pointed to by e that we have requested through malloc().
- Let's try to compile the program and run it.

```
$ gcc -std=c99 employee.c -o employee
$ ./employee < input.txt
Name: Kit
Position: Lecturer
Salary: 20000
Name: Ben
Position: Professor
Salary: 30000
Name: Chim
Position: Lecturer
Salary: 15000
```

Please submit the **employee.c** source file to Moodle.



Talking about *struct*, let us introduce to you one more concept – constructor. The concept of constructor is very important for *class*. You will learn more details in COMP2396 Object-oriented Programming and Java.

struct constructors are a feature of C++ (**but NOT C**) that make initialization of structures convenient. Within a structure type definition, define a constructor in a way that is similar to a function definition, with the following differences.

- The name of the constructor <u>must be the same as the name of</u> the structure type.
- <u>Do not write a return type.</u> The definition starts with the name of the constructor.
- In the body of the constructor, <u>refer to the fields by their names</u>. Do not use a dot. The fields are implicitly those of the structure that is being initialized.



#### Example:

```
struct Student {
    char* name;
    char* id;
    int age;
    Student(char* nm, char* d, int a) {
        name = nm;
        id = d;
        age = a;
    }
};
```

This defines type Student with three fields and one constructor that takes three parameters.

You can use the constructor in three ways.

- Student s("Tim", "3035123456", 20);
  - This is equivalent to:

```
Student s;
s.name = "Tim";
s.id = "3035123456";
s.age = 20;
```

- Student\* s = new Student("Tim", "3035123456", 20);
  - The expression new Student("Tim", "3035123456", 20) creates a new cell in the heap and runs the constructor on that new cell, with the given parameters. Its value is a pointer to the new cell.
- Student("Tim", "3035123456", 20);
  - o To use this way, you have to write the following in your program:

```
Student s = Student("Tim", "3035123456", 20);
```

which is equivalent to:

```
Student s("Tim", "3035123456", 20);
```

A struct can have multiple constructors as follows:

```
struct Student {
   char* name;
   char* id;
   int age;
   Student(char* nm, char* d, int a) {
       name = nm;
       id = d;
       age = a;
   }
   Student(char* nm, char* d) {
       name = nm;
       id = d;
   Student(char* nm) {
       name = nm;
   }
};
```

We call this overloading (functions having the same name but different parameter lists). Again you will learn more details in COMP2396 Object-oriented Programming and Java.

Once again, please be reminded that *struct* constructors are a feature of C++ **but NOT C**. Recall that C++ extends C to include support for Object Oriented Programming and other features that facilitate large software development projects.

Dear Students,

Good job! You have finished 10 self-learning modules in this course! Your effort and hard work are highly appreciated ©.

Please feel free to contact us if you have any doubt when following any of the 10 modules. We are very happy to help you ©!



Wish you all superpass this course!!! Be confident!!! You are now a good programmer and self-learner!!!

T.W. Chim