



速成教育
SPEED UP EDUCATION



CSCA48

Introduction to Computer Science II

导师： VC

2022S CSCA48 Midterm

Unit 1&2 – Memory model and C programming

1.- [5 marks] Consider carefully the following little program and show in the memory diagram below **what variables are allocated** and **their final value (after the program is run)**. Assume the variables are assigned to boxes **in the order they declared**.

```

01 int main()
02 {
03     char *c=NULL;
04     char text[4]={'g', 'c', 'c', '\0'};
05     char w='l';
06     float z=3.1415;
07
08     c=&w;
09     *(c)=text[3];
10     c=&text[2];
11     *(c-2)='b';
12     c=c+1
13     *(c-1)=w;
14     *(c)='d';
15     *(c+4)='*';           // ← a) in question 2
16     c=c-1                 // ← b) in question 2
17     *(c)=65;              // ← c) in question 2
18     // for d) – f) in Q2, assume we would put the corresponding
19     // instruction in line #20 (just below this line)
20
21 }
```

3241 <div></div>	3242 <div></div>	3243 <div></div>	3244 <div></div>	3245 <div></div>
3246 <div></div>	3247 <div></div>	3248 <div></div>	3249 <div></div>	3250 <div></div>

2.- [3 marks] Circle in the space below **any instructions that will likely try to access memory locations your program doesn't own, or cause a memory problem** given the code from Q1. You get .5 for each correctly circled answer, and .5 for each correctly not-circled answer! For d), e), and f) think of each instruction separately - we're seeing what happens if we place the corresponding instruction in line #20 of our program from Q1.

- | | |
|------------------|-------------------------|
| a) Line 15 in Q1 | d) print("%s\n", text); |
| b) Line 16 in Q1 | e) free(c); |
| c) Line 17 in Q1 | f) *(c-6)='\0'; |

Unit 3 – Organizing, Storing, and Accessing Data

For the *Artificial Intelligence* course *CSCD84*, we are implementing a simulation that consists of maze filled with cats, mice, and cheese. The A.I. component consists of implementing algorithms to make the mice behave ‘smartly’: They try to eat cheese while avoiding being eaten by cats. We will call the cats and the mice ‘*agents*’ to indicate they can move around and try to eat something. We will represent them with a CDT:

```
typedef struct agent_struct
{
    double Loc[2];           // Position [x, y] of this agent in the maze
    int hunger;              // Agent's current hunger level
    int type;                // Agent type, 0=cat, 1=mouse, 3=cheese
} Agent;
```

To keep track of the simulation, we also keep a CDT that stores the information from all the agents (there can be up to 10, some cats, some mice, some cheese) that exist at some point in time.

```
typedef struct simulation_struct
{
    int n_agents;            // How many agents in this simulation (<= 10)
    Agent *agents[10];       // Pointers to each of the agents
} Simulation;
```

3.- [5 marks] *Complete the code below.* The code should *initialize a simulation* with *on agent that should be a mouse, at location [2,5], and it just ate so its hunger level is 0*. We are looking for you to show that you have understood how information is organized and know how to access it when we have nested CDTs.

```
int main()
{
    Simulation the_sim;

    // Write the code needed to initialize the_sim to contain 1 agent,
    // and set it up as described above. This should take no more than
    // 6 lines of code (not counting any curly braces you may use).
```

Unit 3 – Linked Lists

Suppose we want to make the simulation more general, in particular we need to be able to handle more than 10 objects (in effect, any number of objects) so we need to keep them in a linked list instead of an array. We have a CDT for the nodes of the linked list:

```
typedef struct agent_node_struct
{
    Agent the_agent;
    struct agent_node_struct *next;
} AgentNode;
```

4.- [1 marks] Suppose we create an AgentNode in main(): AgentNode my_node;
now suppose we call a function and pass it the agent within the agent node: move_agent(my_node.the_agent);
choose the correct option below:

- a) move_agent() will receive a copy of 'the_agent' as a variable it can use
- b) move_agent() will receive a reference (pointer) to 'the_agent' since it's inside a CDT
- c) We can't pass parts of a CDT to move_agent(), we have to pass the whole CDT

5.- [5 marks] **Complete the code below.** It implements the insert function for our linked list of objects. The order of objects in the linked list is **NOT** relevant.

```
AgentNode *insertAgent(AgentNode *head, AgentNode *new)
{
    AgentNode *p=NULL;
    // Inserts the agent whose location is given by the pointer 'new' into
    // the linked list. This has to be done so all mice in the list are stored
    // before any of the cats. As usual, head=NULL if the list is empty.
    // This should take no more than 10-12 lines of code, not counting { }

}
```

Unit 4 – Complexity of list operations

For each of the questions below: **Give a complexity estimate in terms of big O, and briefly justify your answer**

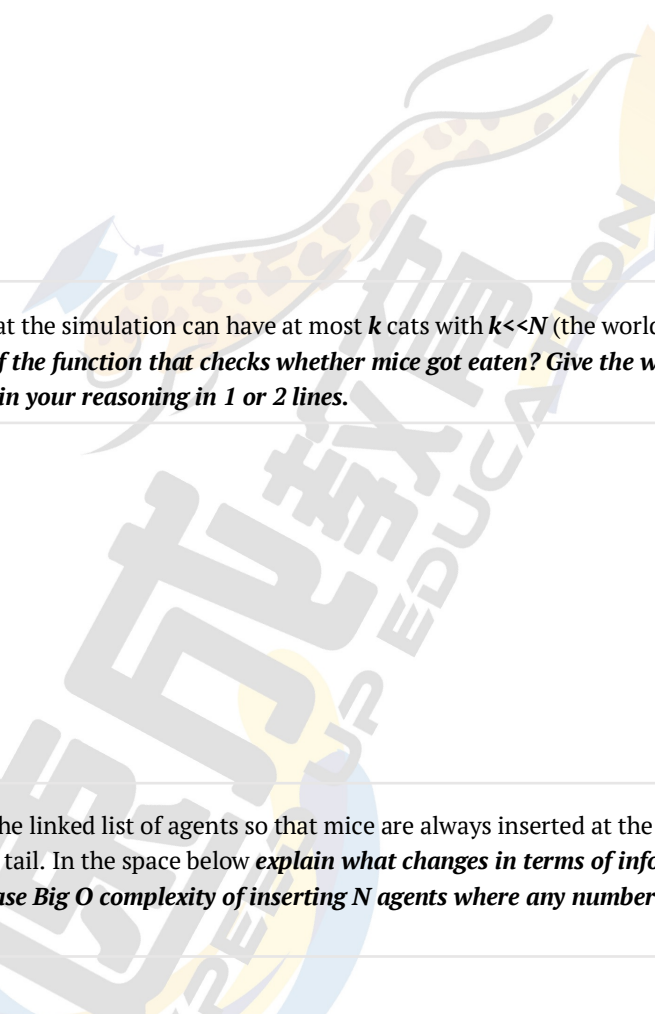
6.- [4 marks] The simulation needs a function to check whether a mouse got eaten by a cat. This happens when they have the same location coordinates. Describe in **very short pseudocode (not C code!)** the process for checking whether or not a mouse got eaten. The give the **Big O worst-case complexity of the function that does the checking if there are N agents, any number of them could be mice, any number of them may be cats.**



7.- [3 marks] Suppose we know what the simulation can have at most k cats with $k \ll N$ (the world seems full of mice!). Does this change the complexity of the function that checks whether mice got eaten? Give the worst-case Big O complexity for this case and explain your reasoning in 1 or 2 lines.



8.- [3 marks] Suppose we design the linked list of agents so that mice are always inserted at the head of the list, and cats are always inserted at the tail. In the space below **explain what changes in terms of information stored in the linked list, and the worst-case Big O complexity of inserting N agents where any number of them are mice and any number of them are cats.**



Unit 1+2+3+4 – Applying what we’ve learned

9.- [3 marks] For A1, you implemented a movie database the allowed you to keep track of the cast for each of the movie stored in it. You also implemented several common queries that directly involved looking at data within the movie’s compound data type. However, typical databases have to answer fairly complex queries that may require looking at information stored in multiple different locations.

Suppose we want to provide a function to **return the box office total for all movies whose cast includes a specific cast member**. Given the structure of the movie database you built for A1, assuming there are N movies, and that each movie’s cast list can have up to M cast members, **what is the worst-case complexity of answering this query?** (Consider here only the number of times cast-member entries have to be searched looking for the query’s requested cast member, do not worry about the operations required to sum up the box office total!) **Give your worst-case estimate in terms of BigO notation, and make sure to briefly explain how you arrived at this estimate.**

10.- [5 marks] The current structure of our database does not help much in terms of answering the query described above. This is a common problem, and the solution to it is to build an **index**. This is just an auxiliary datastructure that makes it faster for us to answer a specific query. In the case above, it looks like we should have an **index on cast-member name** (and for now let’s assume those are unique!).

One proposal for this index is to maintain a **linked list of cast member names, sorted by name**. Inside the node for a given cast member, **we would keep an array with 100 entries**, each capable of storing a reference (a pointer or locker number however you want to think of it) to a movie in our database that this cast member took part in.

In the space below you should be thinking of limitations, of whether or not it really helps us answer queries such as that in 9) more efficiently, and of the impact in terms of extra work (for the developer to implement, and for the computer once it’s running) required to maintain this data structure. **Answer each of the questions in the box using no more than the space provided! - Assume M and N are as defined in 9)**

Advantages of implementing and using this index:

Disadvantages of using implementing and using this index:

Worst-case Big O complexity of answering the query from 9) – with a brief justification

Worst-case Big O complexity of building the index from data in our Movie DB – with a brief justification

Multiple Choice Questions

Consider all options carefully, **and don't forget to record your answers on the bubble sheet - otherwise you will receive a zero. Only record your final answer for each question.**

1.- When we **declare a variable in C** the following thing(s) happen:

- a) Space is reserved in memory for the variable
 - b) The locker for this variable is cleared up
 - c) The name and data type for the variable is set
 - d) Both a) and b)
 - e) Both of a) and c)
-

2.- Which of the things below (if any!) is something **we can't do with a variable** we have declared in our program?

- a) We can find out its locker number
 - b) We can change its data type using casting
 - c) We can change its value using pointers
 - d) We can printf() the variable as a different data type
 - e) Actually, we can do all of a), b), c), and d)!
-

3.- A function has the following declaration: `int copyArrays(int array1[10], int array2[10]);` Without considering **any local variables**. How many boxes will be reserved in memory when the function is called?

- a) 20
 - b) 21
 - c) 2
 - d) 3
 - e) 22
-

4.- From the Unit 2 notes,, which of the options below **is not one reason why different data types exist**

- a) Because data in memory is in binary
 - b) Because the CPU has different circuits for each data type
 - c) To make us think carefully about what data we need to handle, and how it will be used
 - d) So that operations with integers and floats can kept separate
 - e) To allow for checking the consistency of data manipulation in our code
-

5.- Suppose we declare the following CDT:

```
typedef struct midterm_CDT_struct
{
    int x;
    float y;
    char line[5];
} midtermCDT;
```

if in the main() function we declare the following:

```
midtermCDT CDTsRule;
```

How many boxes will be reserved in memory for this?

- a) 7 b) 3 c) 8 d) 4 e) 1
-

6.- Which of the options below is a (are) reason(s) why ADTs are important in CS?

- a) They are abstract, so we can think of them in mathematical terms
 - b) They allow us to think conceptually about data, independently of programming language
 - c) They give us freedom to implement the ADT in a variety of ways
 - d) Both of b) and c)
 - e) Both of a) and c)
-

7.- From the point of view of the amount of time it takes to access information in a linked list, which of the following applications **would not be efficiently solved by using a linked list?** (in particular, consider the order in which information is likely to be accessed!)

- a) Storing the a entries of a matrix we are doing computations with (e.g. for Linear Algebra), when we don't know in advance the size of the matrix we will need.
 - b) Recording all the credit transactions that took place in a supermarket over one week, so that the total amount of sales over this period can be computed.
 - c) Storing product information for an on-line retail website
 - d) Both of a) and b)
 - e) Both of a) and c)
-

8.- In a pinned Piazza post, Paco explained what he thinks is the most likely reason your programs don't pass our testing. Which of the options below was the one thing Paco claimed caused your program to fail?

- a) Because we use a different compiler or operating system than you
 - b) Because of indexing errors when using pointers and arrays
 - c) Because we pick difficult inputs to test your program with
 - d) Because it wasn't properly tested
 - e) Because we check for correctness, not output
-

9.- Which of the complexity classes below corresponds to the hardest problems we can find in computer science?

- a) $O(N^2)$
 - b) $O(N!)$
 - c) $O(N)$
 - d) $O(2N)$
 - e) $O(N \log(N))$
-

10.- Which of the statements below is **not a reason** why we care about thinking in terms of **Big o complexity**?

- a) It gives us information about what algorithms will be faster for every possible input
 - b) The results are valid independently of computing power available
 - c) It let's us compare different algorithms independently of their implementation
 - d) It applies to the large problems, the type of problem we will need to solve if we handle large collections of data (e.g. Google, Amazon, etc.)
 - e) It provides a mathematically rigorous way to analyze and compare algorithmic efficiency
-