CSO-Recitation 06

CSCI-UA 0201-007

R06: Assessment 04 & Strings & Linked list

Today’s Topics

- Assessment 04
- Strings
- Linked list
- Lab2 has bonus

Assessment 04

Q1 C loop

Which loop format checks the condition *before* executing the loop body?

[✓ A. for]

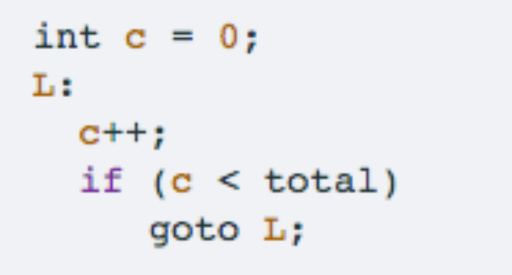
[✓ B. while]

C. do-while execute the statement once before checking the condition

D. all of the above

Q2 Goto

Which is equivalent to the following code fragment containing goto?



```
int c = 0;
L:
  c++;
  if (c < total)
     goto L;
```

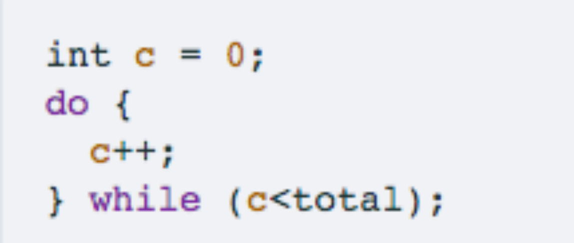
[✓ A.] It's equivalent to a do.. while loop

B. It's equivalent to a while... loop

C. It's equivalent to a for... loop

D. It's equivalent to a if statement

do-while:
The statement is at least executed once



```
int c = 0;
do {
  c++;
} while (c<total);
```

Q3 Variables

Which of the following statements is/are true?

A. The name total at line 8 refers to the global variable total defined at line 3.

B. The name total at line 8 refers to the local variable total defined at line 7.

C. The name total at line 17 refers to the global variable total defined at line 3.

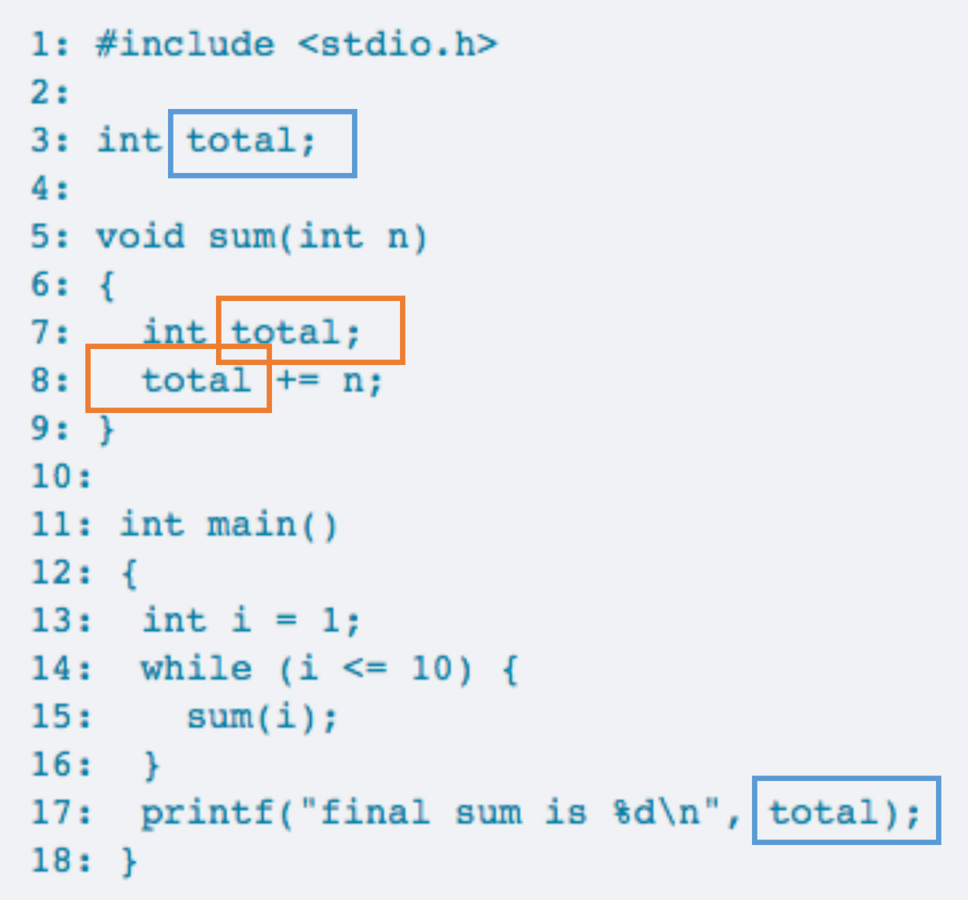
D. The name total at line 17 refers to the local variable total defined at line 7.

E. The global variable total defined at line 3 and the local variable total defined at line 7 share the same underlying storage because they have the same name.

F. The program will cause a compilation error because it defines two variables with the same name total.

G. The global variable total defined at line 3 and the local variable total defined at line 7 are two different variables with separate underlying storage.

Q3 Variables



```c
#include <stdio.h>

int total;

void sum(int n)
{
  int total;
  total += n;
}

int main()
{
  int i = 1;
  while (i <= 10) {
    sum(i);
  }
  printf("final sum is %d\n", total);
}
```

- Global variable:
  - defined outside any function
  - Scope: can be accessed from within any function
  - Storage: allocated upon program start, deallocated when entire program exits
- Local variable:
  - defined inside a function
  - Scope:
    - Within the function/block the local variable is declared
    - Local variables with the same name in different scopes are unrelated
  - Storage: allocated upon function invocation, deallocated upon function return

Q3 Variables

Which of the following statements is/are true?

A. The name total at line 8 refers to the global variable total defined at line 3.

[✓ B.] The name total at line 8 refers to the local variable total defined at line 7.

[✓ C.] The name total at line 17 refers to the global variable total defined at line 3.

D. The name total at line 17 refers to the local variable total defined at line 7.

E. The global variable total defined at line 3 and the local variable total defined at line 7 share the same underlying storage because they have the same name.

F. The program will cause a compilation error because it defines two variables with the same name total.

[✓ G.] The global variable total defined at line 3 and the local variable total defined at line 7 are two different variables with separate underlying storage.

Q4 Followup to Q3

In the example C program of Q3, what is the output of the program?

[✓ A. It always prints 0]

B. It may print some arbitrary value.

C. It always prints 55

[✓ D. None of the above.]

In the absence of explicit initialization,

- Global variables are initialized to 0;
- Local variables have undefined initial values.

Q5 Pointers and arrays

e.g: [“cso”, “recitation”, …, “TA”]
*c == c[0]== “cso”
*(*c+1) == c[0][1] == ‘s’, *(*c) ==‘c’
*(c+1) == c[1] == “recitation”

Given variable definition char *c[10]; what is the type of the expression c[0]+1?

A. char **

[✓ B. char *]

C. char

D. none of the above

char *c[10]:

- c is an array of pointer to char
  - type of c: char **
- c[0] ==*c
  - type of c[0]: char *
- ~~c[0]+1 == *(c+1) ?~~
- c[1] ==*(c+1)
- c[0]+1 == *c+1
  - also pointer arithmetic
  - type of c[0]+1: char *
  - c[0]+1 == *c+1 == &c[0][0]+1 ==&c[0][1]

Q6 Pointers and arrays

Given variable definition char *c[10]; what is the type of the expression c+1?

[✓ A. char **]

B. char *

C. char

D. none of the above

c+1 == &c[1]

Q7 Pointers and arrays

- e.g. c=[’c’, ‘s’, ‘o’, …’r’]
- c[0]==‘c’
- c[1]==‘s’
- c[0]+1==‘d’

Given variable definition char c[10]; what is the type of the expression c[0]+1?

A. char **

B. char *

[✓ C. char]

D. none of the above

char c[10]:

- c is an array of char
  - type of c: char *
- c[0] ==*c
  - type of c[0]: char
- c[0]+1 == *c+1
  - type of c[0]+1: char

Q8 Pointers and arrays

Given variable definition char c[10]; what is the type of the expression c+1?

A. char **

[✓ B. char *]

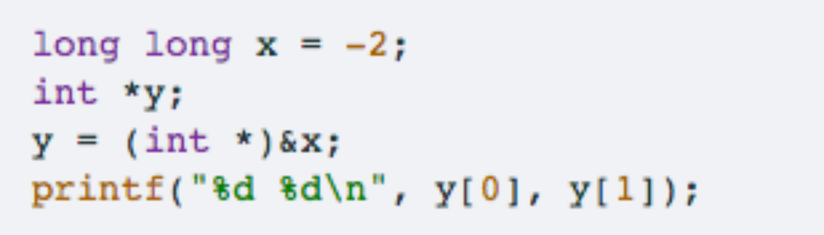
C. char

D. none of the above

c+1 == &c[1]

Q9 Pointer casting

What's the output of the following code fragment (assuming it runs on a 64-bit little endian machine):



```
long long x = -2;
int *y;
y = (int *)&x;
printf("%d %d\n", y[0], y[1]);
```

A. -1 -1

B. -2 -2

C. -1 -2

[✓ D. -2 -1]

E. Segmentation fault

F. None of the above

- long long: 8 bytes
- x is:
  - 0xfffffffffffffffe
- y[0] = *y -> y is a pointer to int
- y[0]=0xfffffffe
- y[1] = *(y+1) -> pointer arithmetic
- y[1]=0xffffffff

Q10 Pointer arithmetic

- *(p+i) == p[i] is often the case
- but it is not always arr[i], it depends on which element your pointer points to

Here's a C code fragment. In order for the above code fragment to output 1 2 10, which of 1 line of code that you should put at Line-3?

A. p[0] = 10;

[✓ B. p[1] = 10;]

C. p[2] = 10;

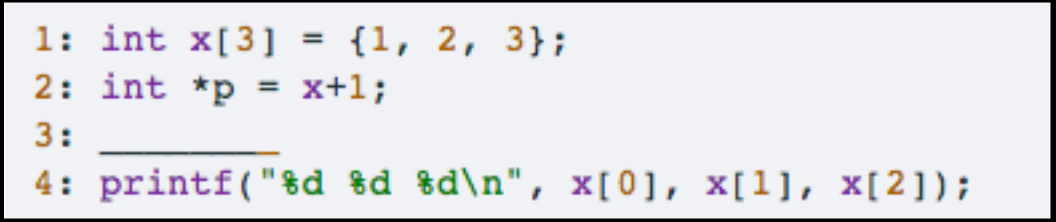
D. *(p) = 10;

[✓ E. *(p+1) = 10;]

F. *(p+2) = 10;

G. p++;

H. p--;



```
1: int x[3] = {1, 2, 3};
2: int *p = x+1;
3: ________
4: printf("%d %d %d\n", x[0], x[1], x[2]);
```

- int *p = x+1;
- p=&x[1], *p=x[1]
- want to set x[2]=10:
  - *(p+1) == x[2]
    - *(p+1)=10
    - *(p+1)==p[1] which is often the case, so:
    - p[1]=10

Strings

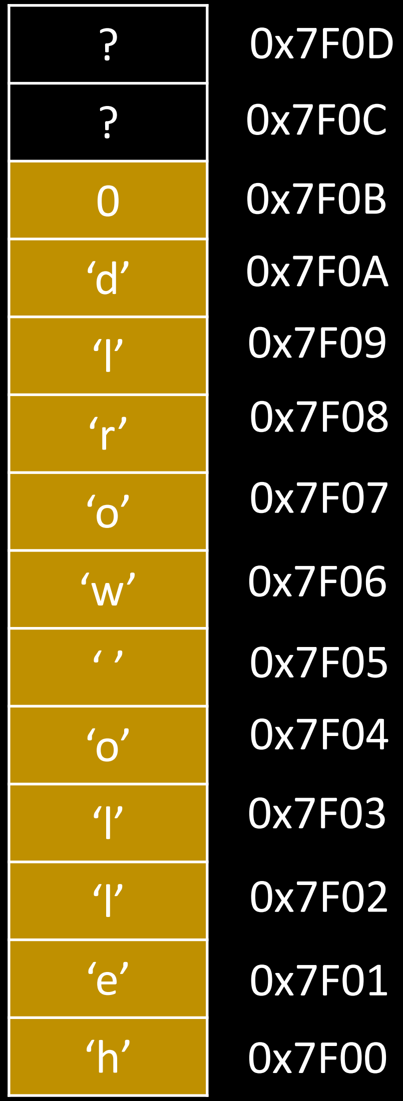
Arrays of chars

What are strings?

- They are arrays of the type *char*, which is typically one byte
- Char literals are in single quotes ‘ ’
- String literals are in double quotes “ “
- Unlike other arrays, strings have a way of knowing the length even at runtime
  - Strings are stored with the last byte set to 0 (or ‘\0’)
    - C strings are called “null terminated”
    - So you can find the length by looping over the string, keeping a counter, and stopping when you find a char equal to zero
  - There is also a standard library function for this, *strlen*

Defining a string

- char *arr = “hello world”;
- char arr[11] = “hello world”;
- The literal “hello world” includes the null-terminator.



?	0x7F0D
?	0x7F0C
0	0x7F0B
‘d’	0x7F0A
‘l’	0x7F09
‘r’	0x7F08
‘o’	0x7F07
‘w’	0x7F06
‘ ’	0x7F05
‘o’	0x7F04
‘l’	0x7F03
‘l’	0x7F02
‘e’	0x7F01
‘h’	0x7F00

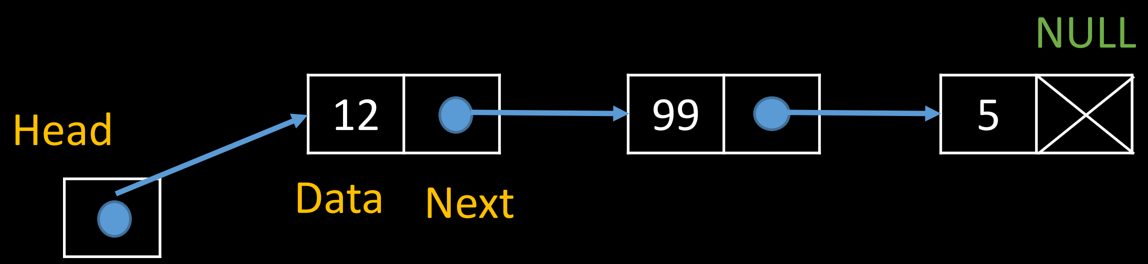
Array of pointers: argv

- argv is an array of strings (pointers to char)
  - the strings are your arguments
  - argv[0] is the name of the executable file
- argv has argc many elements

Linked list

A linear data structure

Why linked list?



- Like arrays, Linked List is a linear data structure.
- Unlike arrays, linked list elements are not stored at a contiguous location; the elements are linked using pointers.
- Arrays have limitations:
  - The size of the arrays are fixed
  - Inserting (Deleting) a new element in an array of elements is expensive
    - because the room has to be created for the new elements and existing elements have to be shifted.

Advantages and Drawbacks

- Advantages over arrays:
  - Dynamic size
  - Ease of insertion/deletion
- Drawbacks:
  - Random access is not allowed
    - We have to access elements sequentially starting from the first node. (Traverse)
  - Extra memory space for a pointer is required with each element of the list.
  - Not cache friendly
    - Since array elements are contiguous locations, there is locality of reference which is not there in case of linked lists.

Linked list

- A linked list is represented by a pointer to the first node of the linked list
  - It is called the *head*
  - If the linked list is empty, then the value of the head is NULL
- Each node in a list consists of at least two parts:
  - data
  - Pointer (or Reference) to the next node
- In the case of the last node in the list,
  - the next field contains NULL - it is set as a null pointer.
- In C, we can represent a node using struct
  - nodes are defined as (e.g.) node using *typedef*
  - *node *head*

Initialize the linked list

- The list is initialized by creating a *node *head* which is set to NULL
- The variable *head* is now a pointer to NULL, but as nodes are added to the list, *head* will be set to point to the first node
- In this way, *head* becomes the access point for sequential access to the list.

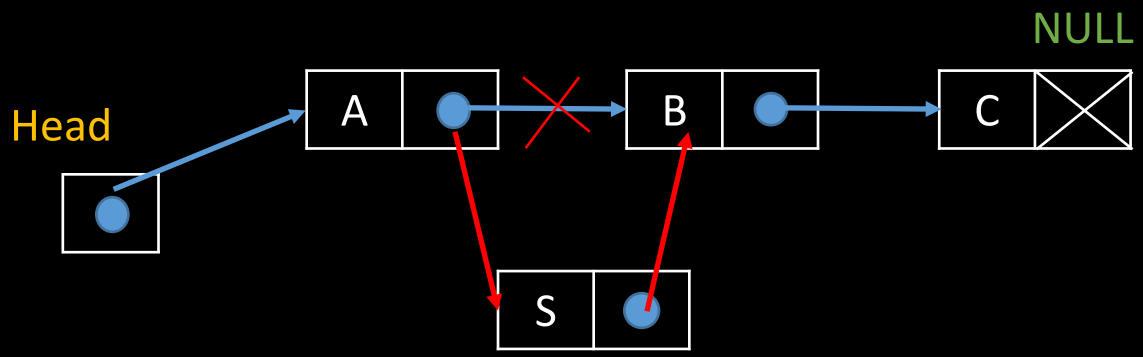
Linked list

- Linked list insertion
- Linked list Deletion
- Search an element in a linked list
    - Iterative and Recursive
- Traverse a linked list
- Find length of a linked list
- …

Linked list

- In class, we pass the header pointer,
    - ask it to return a new head
    - the caller is responsible for updating it itself
- In lab-2, we pass a pointer to pointer parameter (pointer to the head pointer),
    - to allow changing the head pointer directly instead of returning the new one
    - note that there’s no return value; It’s not needed.

Inserting a node



- How can we insert a node in a sorted linked list?
  - Insert a node at the front of the linked list
    - insert_front
  - Insert a node after a given node
    - Think: how can I know my S should be inserted between A and B?
    - If by comparing A and S I know S should be at the position after A, then how can I know S should be after B or between A and B?
  - Insert a node at the end of the linked list

Dynamic memory allocation

- Each time you need to manually allocate data, use *malloc*
  - void *malloc(size_t size);
- If you need to manually de-allocate
  - void free(void *ptr);

More on linked list

- Implement a hash table
  - see clear instructions on our website lab-2 page
- A hash table is an array of linked lists with a hash function
  - A hash function basically just takes things and puts them in different “buckets” (hash table’s array of entries)
  - Each “bucket" just points to a linked list here