# CSE333 SECTION 3

# Malloc

- Allocate memory block in heap
- void* malloc (size_t size);

- size: Size of the memory block, in bytes.

- On success, a pointer to the memory block allocated by the function.
- If the function failed to allocate the requested block of memory, a *null pointer* is returned.

# String in C

- Represented an array of characters:
- char label[] = "Single”;
- ---------------------------
- | S | i | n | g | l | e | \0 |
- ---------------------------

- A character array can have more, as below:
- char label[10] = "Single";
- giving an array that looks like:
- ---------------------------------------
- | S | i | n | g | l | e | \0 |  |  |  |
- ---------------------------------------

# <string.h>

- size_t strlen ( const char * str );
- **Get string length**

- char * strcpy ( char * destination, const char * source );
- **Copy string**

- char * strcat ( char * destination, const char * source );
- **Concatenate strings**

# Exercise

```
#include <stdio.h>

void myStrcat(char *a, char *b)
{
    int m = strlen(a);
    int n = strlen(b);
    int i;
    for (i = 0; i <= n; i++)
       a[m+i]  = b[i];
}

int main()
{
    char *str1 = "Geeks ";
    char *str2 = "Quiz";
    myStrcat(str1, str2);
    printf("%s ", str1);
    return 0;
}
```

We want the program to print "Geeks Quiz", does it?

# Exercise

What is the output of following program?

```
# include <stdio.h>

int main()
{
   char str1[] = "GeeksQuiz";
   char str2[] = {'G', 'e', 'e', 'k', 's', 'Q', 'u', 'i', 'z'};
   int n1 = sizeof(str1)/sizeof(str1[0]);
   int n2 = sizeof(str2)/sizeof(str2[0]);
   printf("n1 = %d, n2 = %d", n1, n2);
   return 0;
}
```

# Solution

N1=10 n2=9

# Exercise

What does the following fragment of C-program print?

```
char c[] = "GATE2011";
char *p =c;
printf("%s", p + p[3] - p[1]) ;
```

# Solution

2011

# Exercise

```
#include<stdio.h>
int main()
{
    char str[] = "GeeksQuiz";
    printf("%s %s %s\n", &str[5], &5[str], str+5);
    printf("%c %c %c\n", *(str+6), str[6], 6[str]);
    return 0;
}
```

- A Runtime Error
- B Compiler Error
- C uiz uiz uiz u u u
- D Quiz Quiz Quiz u u u

# Solution

D

# Exercise

Assume that a character takes 1 byte. Output of following program?

```
#include<stdio.h>
int main()
{
    char str[20] = "GeeksQuiz";
    printf ("%d", sizeof(str));
    return 0;
}
```

# Solution

20

# Exercise

Predict the output of following program, assume that a character takes 1 byte and pointer takes 4 bytes.

```
#include <stdio.h>
int main()
{
    char *str1 = "GeeksQuiz";
    char str2[] = "GeeksQuiz";

    printf("sizeof(str1) = %d, sizeof(str2) = %d",
          sizeof(str1), sizeof(str2));

    return 0;
}
```

# Solution

4, 10

# Exercise: strcat333

Function prototype:

void strcat333(char *s1, char *s2, char **result);

Concatenate s1 and s2, dynamically allocating the result and returning it through the output parameter “result”

EX:

s1 = “CSE”;

s2 = “333”;

result is the string “CSE333”

# Structs without typedef

struct <struct name> {

<field>

. . .

<field>

};

Declaration:

struct <struct name> <instance name>;

struct <struct name><instance name> =

{.<field name> = <value>, …, .<field name> = <value>};

# Structs with typedef

typedef struct <struct name (optional)> {

<field>

. . .

<field>

} <typedef name>, …, <typedef name>;

Declaration:

<typedef name> <instance name>;

<typedef name> <instance name> =

{.<field name> = <value>, …, .<field name> = <value>};

# Struct usage

- Use “.” (dot) to refer to fields in a struct
- Use “->” (arrow) to refer to fields through a pointer to a struct

```
typedef struct point {
   int x, y;
} Point, *PointPtr;

int main(…) {
   Point p = {.x = 5, .y = 7}
   p.x = 4;
   PointPtr p_ptr = &p;
   p_ptr->y = 4;
}
```

# LinkedList

```
// linked list node for a list of c-strings
typedef struct node {
 char * data; // string data in this node
 struct node * next; // next node or NULL if none
} Node;
```

# Exercise: Find the bug

```
// print strings in list that starts at head
void prlist(Node * head) {
 Node * p = head;
 while (p != NULL) {
 printf("%s\n", p->data);
 p = p->next;
 }
}
// add x to front of strlist and return pointer to new list head
Node * push_node(Node x, Node * strlist) {
 x.next = strlist;
 return &x;
}
```

```
// link two nodes together as a list and then print the list
int main(int argc, char ** argv) {
 Node n1;
 Node n2;
 Node * list = NULL; // head of linked list or NULL if empty
 // copy "world" to first node and push onto front of list
 strcpy(n1.data, "world");
 list = push_node(n1, list);
 // copy "hello" to second node and push onto front of list
 strcpy (n2.data, "hello");
 list = push_node(n2,list);
 // print list
 prlist(list);
 return EXIT_SUCCESS;
}
```

# Solution

1. Function push_node returns the address of a local variable (x) that no longer exists after the function returns. The best fix that matches the intent of the original code is to change push_node to use a pointer for its first parameter

2. In the strcpy function calls (e.g., strcpy(n1.data, “hello”);) the data pointers are not initialized and do not point to a character array where a copy of the string can be stored.

# Exercise: Deep Free

```
typedef struct node {
 char * data; // string data in this node
 struct node * next; // next node or NULL if none
} Node;
```

Free a Linked List whose nodes are defined above. Assume that both the Nodes and the data within them have been dynamically allocated.

Function Prototype:

```
void FreeLinkedList(Node *lst);
```