Tutorial 7
Linked lists

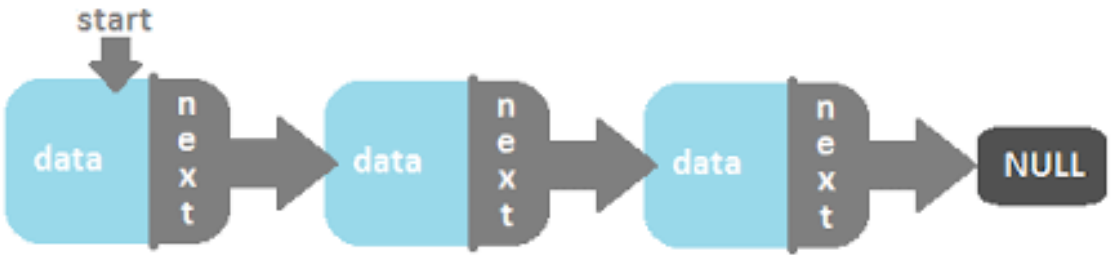
Jiawei Li (Michael)

Office hours: Wednesday 2:00-4:00pm

Office: PMB426

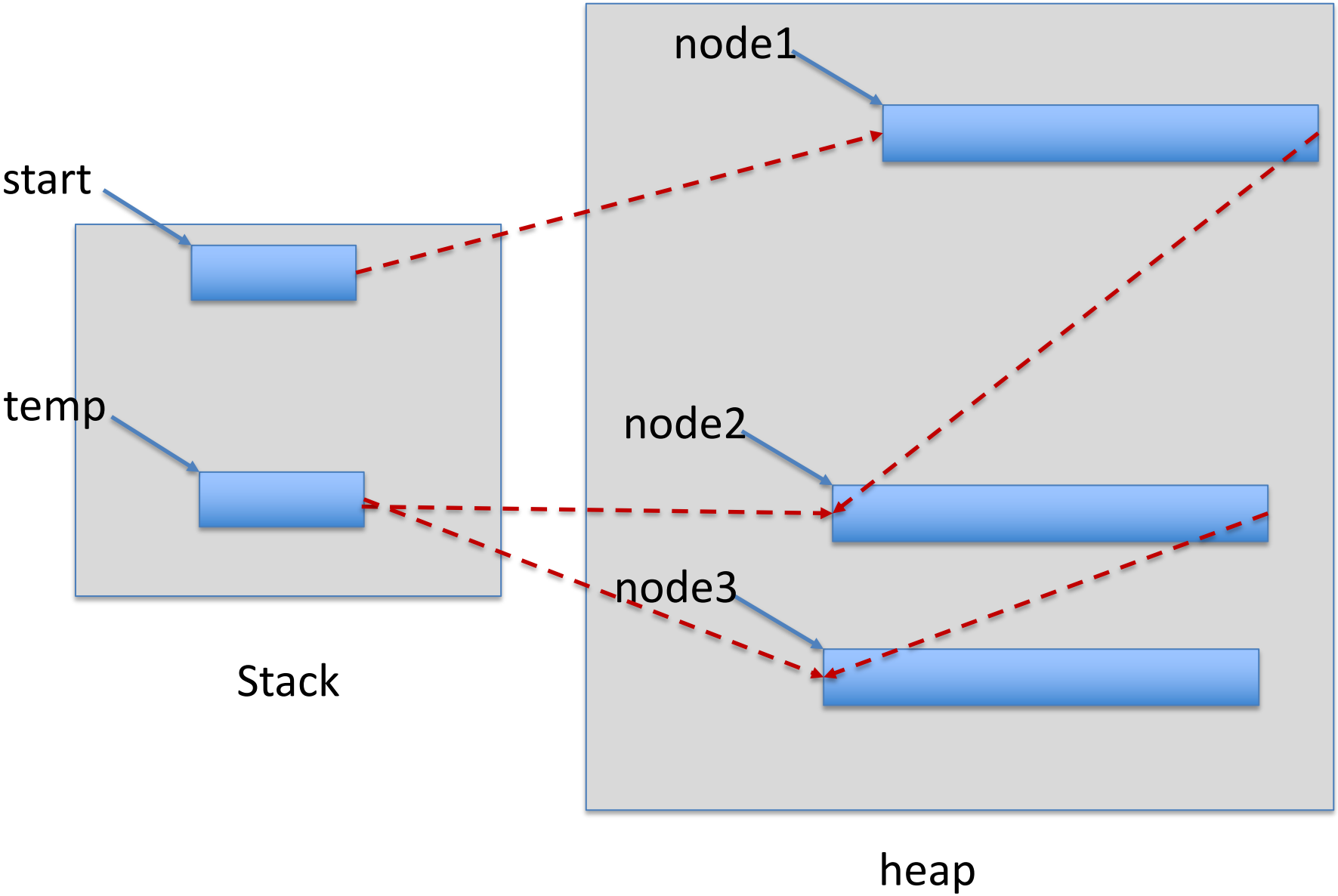
Email: jiawei.li@nottingham.edu.cn

Singly linked lists

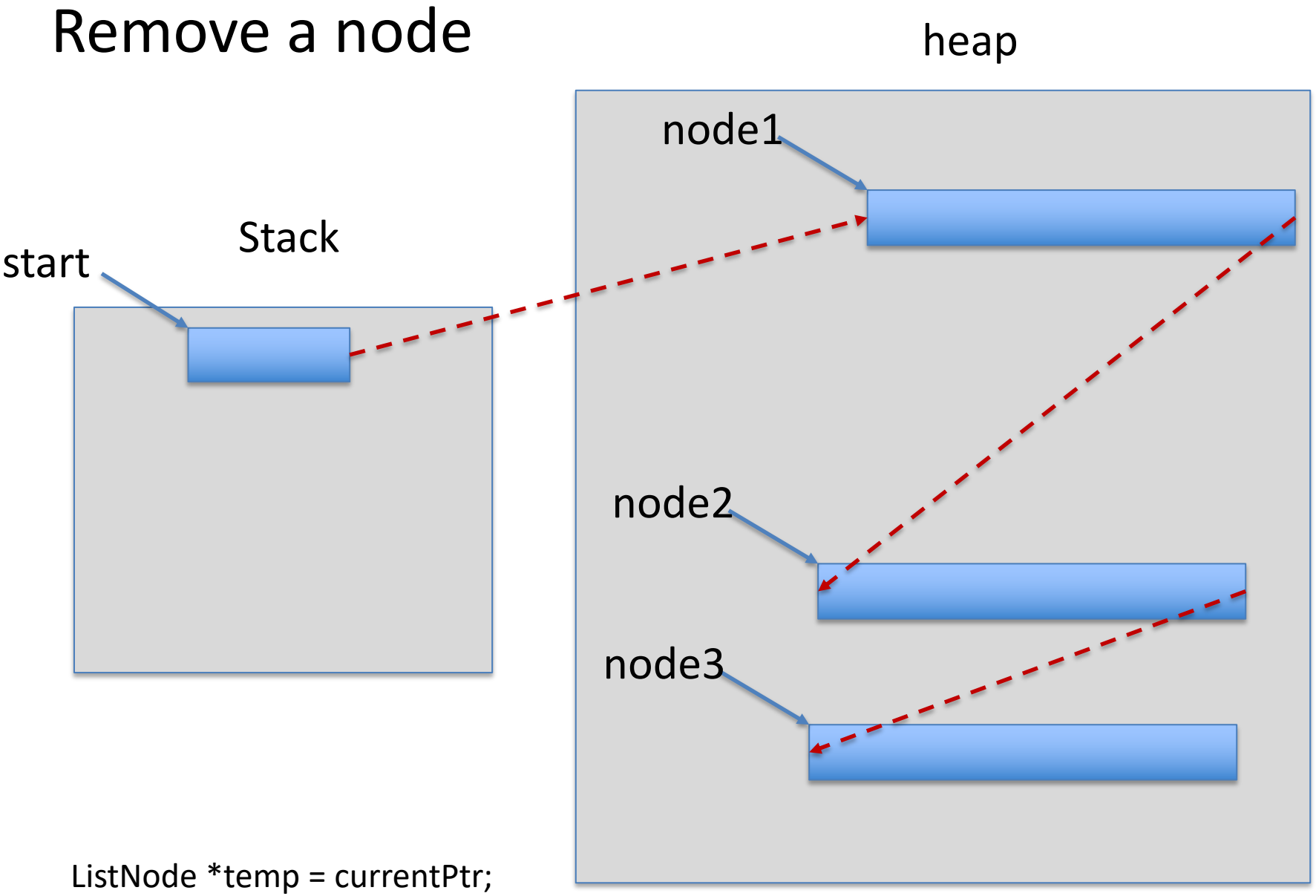


Doubly linked lists

Create a list



Remove a node




```
ListNode *temp = currentPtr;
previousPtr->nextPtr = currentPtr->nextPtr;
free(temp);
```

Lists

```
struct listnode {
        int ID;
        char name[20];
        struct listnode *nextPtr;
};

typedef struct listnode Listnode;
typedef Listnode *ListNodePtr;

int main(void)
{
        ListNodePtr start = NULL;
        …
        insert(&start, value, string);
        …
}
```

Global

Local

Global and local variables

```
#include <stdio.h>

void func1(void);
void func2(void);

int a;

int main(void)
{
	a = 1;

	func1();
	printf("%d\n", a);

	func2();
	printf("%d\n", a);

	return 0;
}
```

```
void func1(void)
{
	int a = 2;
	a++;
}

void func2(void)
{
	a++;
}
```

Process structure in memory

Stack

Data area that grows downwards towards the heap

LIFO data structure, for local variables and parameters

Heap

Data area that grows upwards towards stack

Specially allocated memory (malloc, free, …, probably new, delete)

Data and BSS (uninitialised data) segment

Read-only:	Constants	String literals
Read/write:	Global variables	Static local variables

Code (or text) segment

The program code

Lists

```
struct listnode {
        int ID;
        char name[20];
        struct listnode *nextPtr;
};

typedef struct listnode Listnode;


//list of prototypes of functions
void insert(ListNode **sPtr, int value,
char *string);
int delete(ListNode **sPtr, int value,
char *string);
…
…
```

```
int main(void)
{
        ListNode *start = NULL;
        …
        insert(&start, 1, “Paul”);
        …
        delete(&start, 2, “Michael”);
}

void insert(ListNode **sPtr, int value,
char *string)
{
        …
}
int delete(ListNode **sPtr, int value,
char *string)
{
        …
}
```

```
void insert_element(ipalist_t **start, int val)
{
  if(*start == NULL)                          // This means that the list is empty
  {
    ipalist_t *n = malloc(sizeof(ipalist_t));  // Create the first node. The address of this
    n->value = val;                            node is extremely important!!!
    n->next = NULL;
    *start = n;
  }
  else                                        // The list is not empty, so we add in a new node
  {
    ipalist_t *cur = *start;                  // Create a copy of *start because we do not
    while(cur->next != NULL)                  want to make any change to *start. Why?
    {
       cur = cur->next;
    }
    ipalist_t *n = malloc(sizeof(ipalist_t));
    n->value = val;
    n->next = NULL;
    cur->next = n;
  }
}
```

Frequent mistakes

- Type of data does matter.
- Do we have to use pointers to pointers?
- How to create the first node?
- Remove a node or remove the list?

```
struct student_list
{
    int ID;
    char name[20];
    int *next;
};  // Anything wrong in the code?
```

Frequent mistakes

```
struct student_list* LinkedList()
{
    struct node *root;
    struct node *conductor;
    root = malloc(sizeof(struct root));
    …
}  // Anything wrong in the code?
```

```
void list_print(student_list** head)
{
    while(*head!=NULL)
    {
        printf(“%d”, *head->ID);
        *head = (*head)->next;
    }
}  // Anything wrong in the code?
```

Exercise for you

A program with the name of list.c is available on the Moodle. It allows the user to create a simple char list.

Your task is to write a function reverseList() which creates a new list that contains all the nodes of the original list in reverse order. For example, if the original list contains "a f f g k", the new list should be "k g f f a".

Declaration of the function is:

```
void reverseList(ListNode **newList, ListNode *currentList);
```