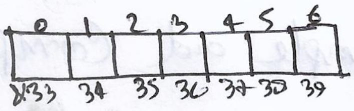
7 36 18 25 22 12

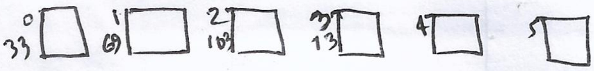
Linked List

Motivation ?

array  connective, size fixed, space ~~usage~~ allocation optimize

এমন situation → কত size লাগবে জানি না। solution: যতটুকু প্রয়োজন ততটুকু allocate করো

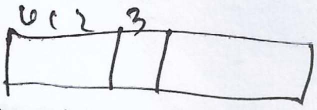
new challenge



~~বিচ্ছিন্ন~~ but connected
ছড়ানো ছিটানো.

compound data type

value
pointer to next
pointer to previous

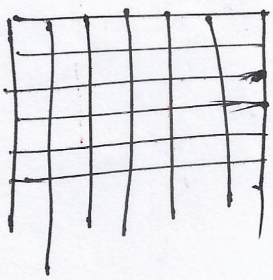


আগে পরে

চিনে নেওয়া যায় যেহেতু পাশাপাশি থাকে.

but ? ← [ ] → ?

Memory
2D grid



Stack
Heap
region

vs
stack
queue

Data type

first address + n × size of data type

for accessing array

---

scope

Linked List → Data Structure গঠিত/অগঠিত Data কিভাবে রাখব 3 রকমের!

Basic

char → 1 byte → -128 ~ 127

Short → 2 byte → -32768 ~ 32767

int → 4 byte → -2147483648
2147483647

64x long → ~~8 byte~~ 32x

4	8
8	8

64x

long long

float → 4 byte

double → 8 byte

User Defined

Simple and Compound

ওটুকু জন্য int a ; int a = 52;
কাটিছে use করতে দিবে না. → int arr[5]; int arr[] = {1,2,3,4};
int *

char ch = 'p'; char str[] = "abcde"
char *str = "abcde"
zero idx address.

Stack allocation

---

heap allocation int *b = (int *) malloc (4); byte
" sizeof(int)
n * sizeof(int) → equivalent to array.

float *d = (float *) malloc (n * size of (float));

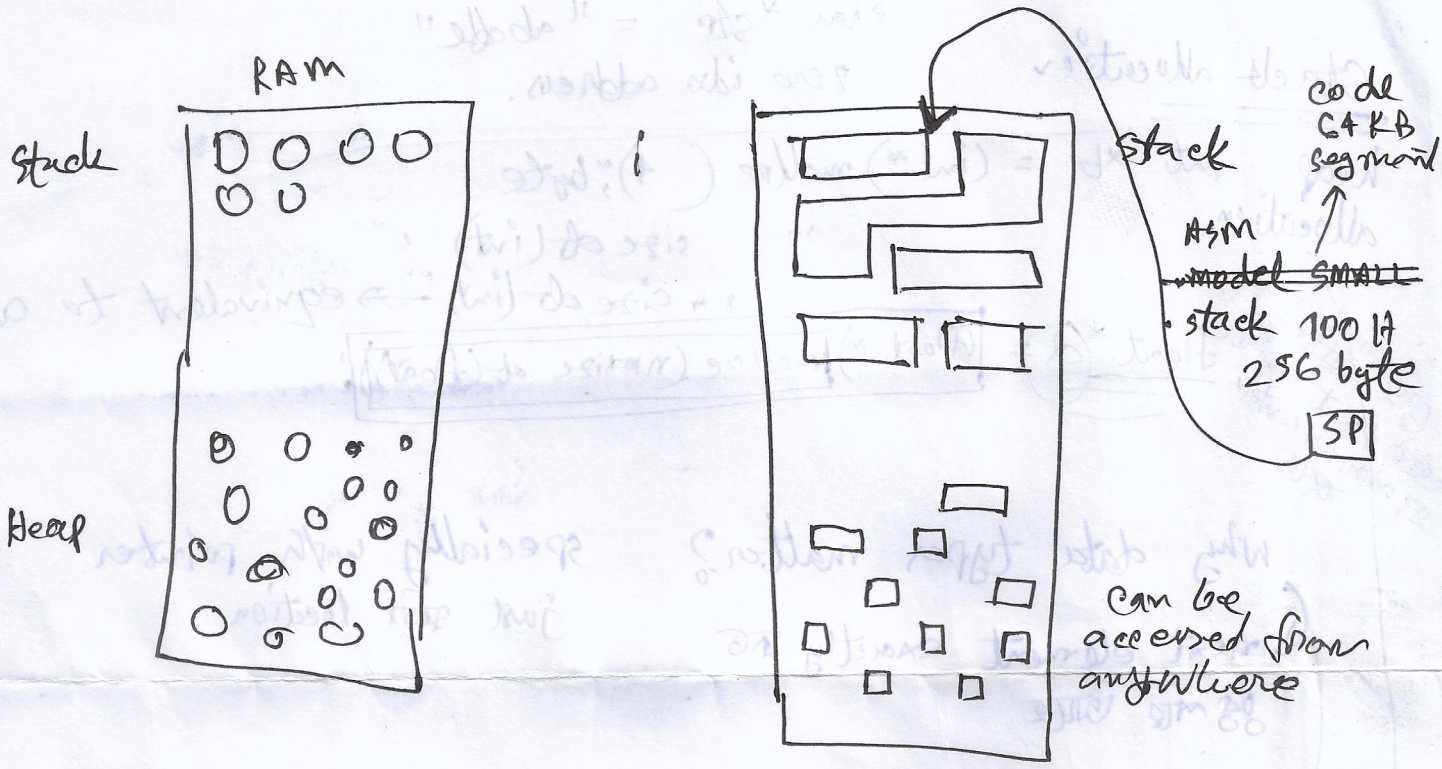
এটা pointer of float variable এর address contain করে

Why data types matter? specially with pointer
just একটা location

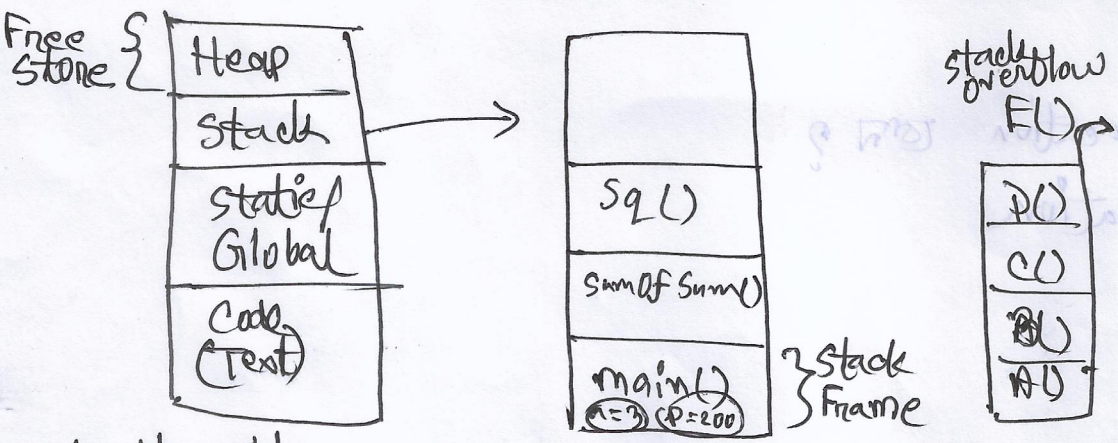
→ next element exactly কত দূরে আছে

→ compound data type (struct)
এ কে কতটুকু জায়গা দখল করে আছে

→ Heap allocation কেন?
→ motivation



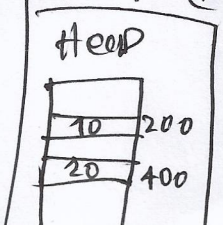
Application Memory



c: malloc, calloc, realloc, free } func
default void

c++: new, delete } operator

```
int a = 3
int *p = (int*) malloc (sizeof(int));
*p = 10;
p = (int*) malloc (sizeof(int));
p = 20
```

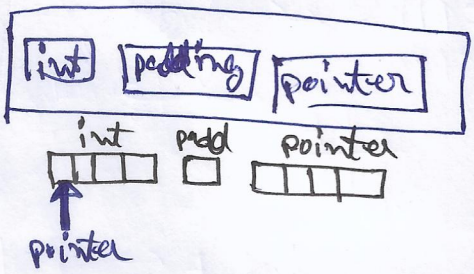


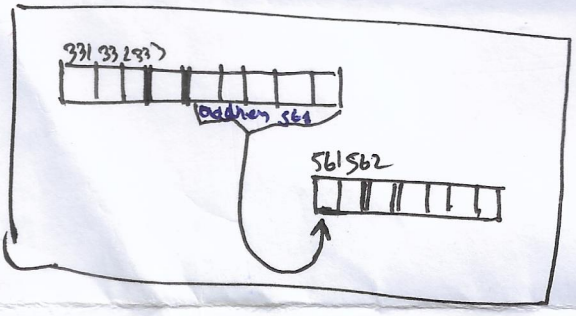
node

```
struct linkedlist {
    int data;
    struct linkedlist *next;
} node;
```

node

struct linkedlist → compound data type






```
#define true 1
#define false 0
struct LinkedList{
    int data
};
typedef struct linkedlist node;
typedef int songkhar;
```

Linked List Code 1

```c
#include <stdio.h>
#include <stdlib.h>

struct node
{
    int data; // int type er ekta address store korbe
    struct node* next; // struct type er ekta address store korbe
};


int main(void) {
    struct node *n1, *n2, *n3;

    n1 = (struct node*)malloc(sizeof(struct node));
    n2 = (struct node*)malloc(sizeof(struct node));
    n3 = (struct node*)malloc(sizeof(struct node));

    n1→data = 10;
    n2→data = 20;
    n3→data = 30;

    n1→next = n2;
    n2→next = n3;
    n3→next = NULL;

    // printf("size of node = %d\n", sizeof(struct node));
    printf(" n1→data = %d \n", n1→data);
    printf(" n1→next→data = %d \n", n1→next→data);
    printf(" n1→next→next→data = %d \n", n1→next→next→data);


    // better print process
    struct node* tracker = n1; // ekta tracker variable jar value bar bar change hobe
    while (1) { // koto gulo node ache, jani na, colte thakbo, ek time e jeye thambo
        printf("<%d>", tracker→data);
        tracker = tracker→next;
        if (tracker == NULL) {
            break;
        }
    }

    // further better
    while (tracker != NULL) {
        printf("<%d>", tracker→data);
        tracker = tracker→next;
    }
```

```
    struct node *head, *prev;

    // better node creation process
    // create 30 nodes which will contain values from 1 to 30
    int counter = 1;
    while (1) { // note: ei while loop for diye o lekha jay
        struct node* newNode = (struct node*)malloc(sizeof(struct node));
        // counter er value node id te store korbo
        newNode→data = counter;
        newNode→next = NULL;

        // 1→2→3→4→5→....→28→29→30
        // 3 ta case hote pare
        if (counter == 1) { // first case
            head = newNode; // head er address store korte hobe // head haray gele bipod
            prev = newNode; // prev node is the tracker node
        }
        else { // second case (any middle node)
            prev→next = newNode;
            // ager loop er por ei loop e same name e ekta n toiri hosse
            prev = newNode;
        }

        // 3rd case (last node, and, terminate looping)
        if (counter == 30) {
            break;
        }

        counter++;
    }
    // পূর্বের নোডের সাথে বর্তমান নোড কানেক্ট করার জন্য আমাকে দুইটা নোডের (পূর্ব ও বর্তমান) লোকেশন হাত এ
রাখা লাগবে (ভ্যারিএবল এ
    // রাখা লাগবে), নইলে এড্রেস হারিয়ে ফেললে আমি আগের নোডের সাথে বর্তমান নোড লিংক করতে পারব না
}
// EOF
```

Linked List Code 2

```
#include <stdio.h> // WORKS
#include <stdlib.h> // BASIC SINGLY LINKED LIST
struct node
{
    int data;
    struct node* next;
};
typedef struct node NODE;

void PrintNodes(NODE* p) {
    while (p =≠ NULL) {
        printf("%d ", p→data);
        p = p→next;
    }
}


void PrintNodesForFun(NODE* head) {
    for (NODE* i = head; i =≠ NULL; i = i→next) {
        printf("%d ", i→data);
    }
} // this is valid


int main(void) {
    int count = 0, i = 0;
    NODE *new_node_in_heap, *head;

    // p1 = (struct node *)malloc(sizeof(struct node));
    // p2 = (struct node *)malloc(sizeof(struct node));
    // p1→next = p2;
    // p3 = (struct node *)malloc(sizeof(struct node));
    // p2→next = p3;
    // ...
    // pN = (struct node *)malloc(sizeof(struct node));

    printf("How many values do you need to store? ");
    scanf("%d", &count);


    // 0→1→2→3→...→count
    NODE* prev; // লুপিং এর জন্য
    for (int i = 0; i < count; i++) {

        int new_node_data = i;
        // printf("enter node data: ");
        // scanf("%d", &new_node_data);

        new_node_in_heap = (NODE*)malloc(sizeof(NODE));
```

```
        new_node_in_heap→data = new_node_data;
        new_node_in_heap→next = NULL;

        if (count == 0) {
            head = new_node_in_heap;
            prev = new_node_in_heap;
        }
        else {
            // still prev e ache ager ta
            prev→next = new_node_in_heap;
            // শোনো প্রথম বক্স, তোমার মেমরির নেক্সট  অংশে সেকেন্ড/পরেরজন কে চিনে রাখো

            prev = new_node_in_heap;
            // কারেন্ট, তুমি এবার পরের জনকে ধর
        }
    }

    PrintNodes(head);

    // EOF
} // DONE
```

Personal Lined List Header Code

```
#include <limits.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define true 1
#define false 0

struct linked_list {
  int number;
  struct linked_list *next;
  struct linked_list *prev;
  char node_name[6];
};
typedef struct linked_list node;
```

Linked List Code 3 (Traverse Reversely)

```
#include "LinkedListHeader.h"

void PrintLinkedList(node *vertex) {
    int counter = 0;
    while (vertex != NULL) {
        // NOT //while (vertex→next != NULL) { // NOT //
```

```
        printf("%d.[%d]\n", counter + 1, vertex→number);
        vertex = vertex→next; // for last node, this expression will copy NEXT's
        counter++;
    }
} // works

void Traverse(node *n) {
    if (n→next == NULL) { // base case
        printf("%d\n", n→number);
        return;
    }
    Traverse(n→next); // using the concept of recursion
    printf("%d\n", n→number);
    //return;
}

int main(void) {
    node *head, *tail, *curuent;
    int n = 0;
    scanf("%d", &n);

    for (int i = 0; i < n; ++i) {
        node *n = (node *) malloc(sizeof(node));
        n→next = NULL;
        scanf("%d", &n→number);
        if (i == 0) {
            head = n;
            curuent = n;
        } else {
            curuent→next = n;
            curuent = n;
        }
        if (i == (n - 1)) {
            tail = n;
        }

    }

    PrintLinkedList(head);
    Traverse(head);
    // end
} // done
```