Dynamic Memory Allocation Continued

free(): Memory De-allocation

- Used to dynamically de-allocate memory
- Allocated using functions malloc() and calloc()
- Is not de-allocated on their own
- Used to reduce wastage of memory

```
free(ptr);
```

realloc(): Re-allocation

- Used to dynamically change the memory allocation of a previously allocated memory
- Useful when memory allocated with malloc() / calloc() is insufficient

```
ptr = realloc(ptr, newSize);
```

Advantages

- Flexibility
- No Exact Memory Requirements
- Efficient Memory Usage
- Memory Reusability

Disadvantages

- Security Concerns
- Memory Wastage
- Memory Leak
    - Fragmentation
    - Allocate Too Much

DMA Pointers and Typecasting

- Declared a variable
- Declare a pointer
- Assign pointer to point to the variable
- Allocating memory requires comfort in pointers
- Point to an "unamed" memory location
- Access memory via pointers
- Typecasting not necessary, **however** I recommend always typecast an malloc(...) just to be sure there is no ambiguity.

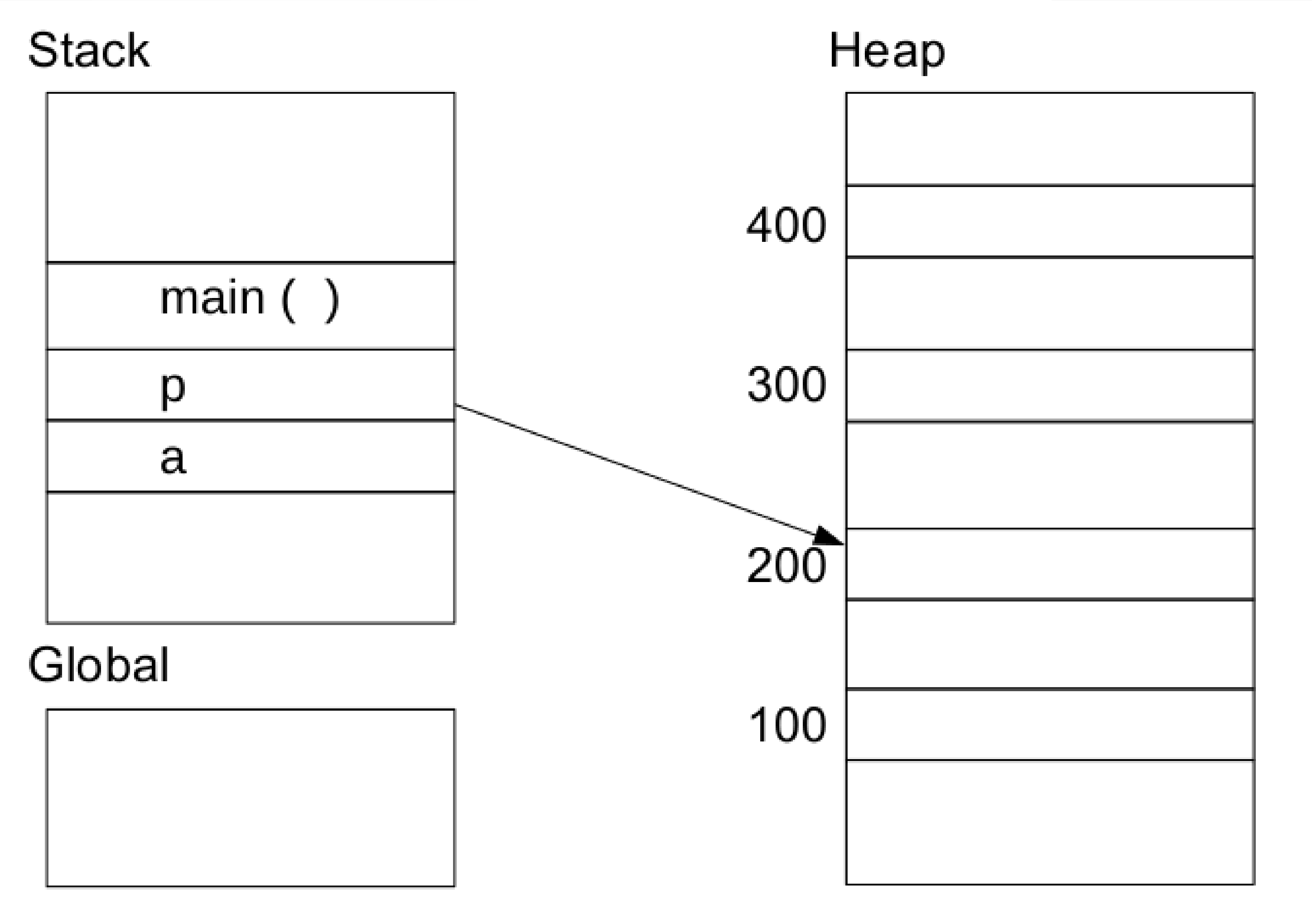
Advantages to casting

- May allow a C program or function to compile as C++
- Allows correcitons for pre-1989 versions of malloc
- Help developer identify inconsistencies in type sizing

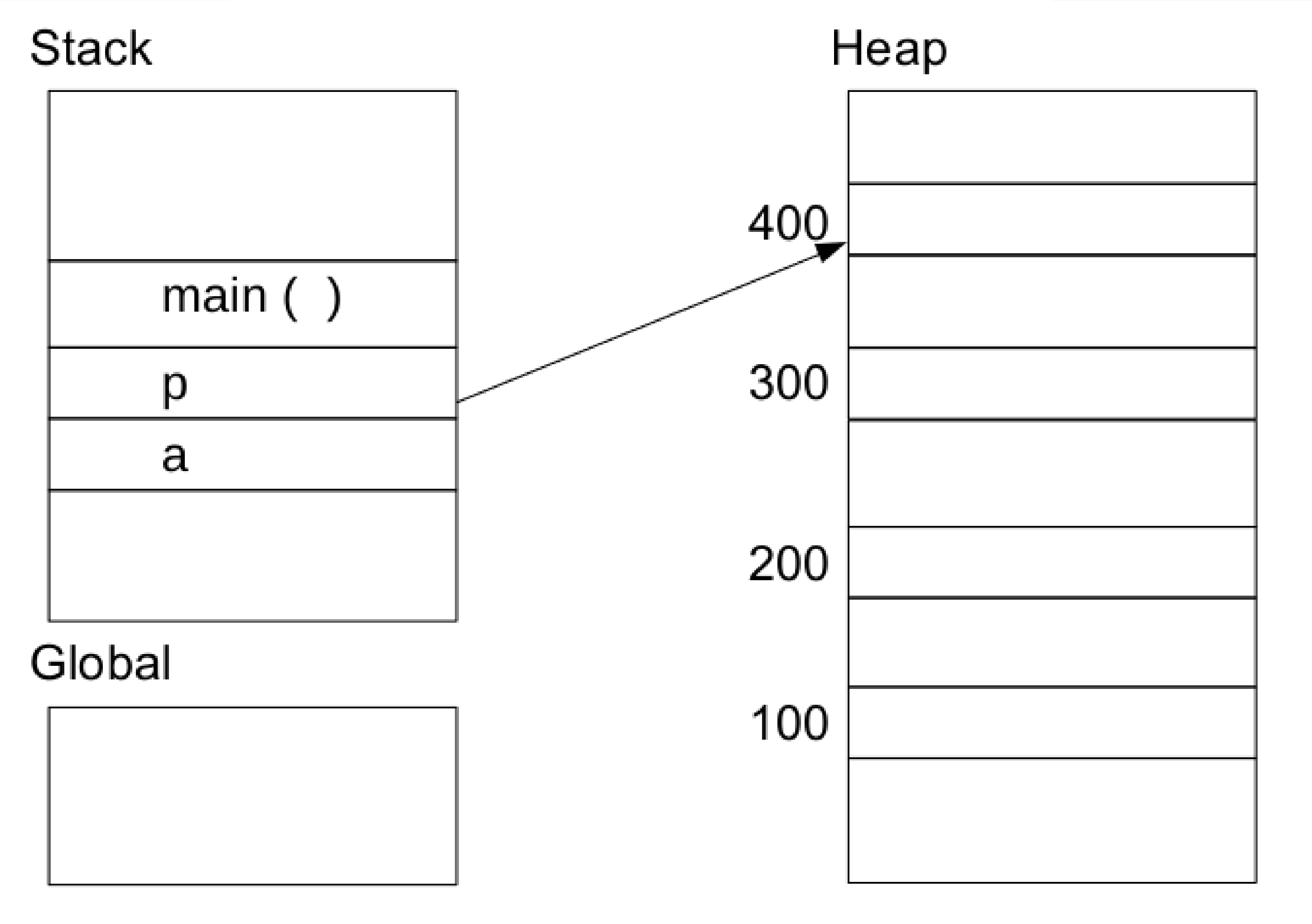
Disadvantages to casting

- C standard, the cast is redundant
- May mask failure to include the header stdlib.h
- C90 standard requires C compiler malloc returns an int
- Could stop a diagnostic run by the C90 standard
- Going from 32- bit to 64-bit systems problem can occur

Dynamic Allocation



Dynamic Allocation



Dynamic Memory Allocation in Arrays

- Useful the size of an array is not known
- Can allocate memory for an array during runtime

```c
int *arr, i, n;
printf("Enter size: ");
scanf("%d", &n);
arr = (int*) malloc(n * sizeof(int));

// accessing array elements
for(i = 0; i < n; ++i) {
    scanf("%d", arr + i);
}
```

Dynamic Memory Allocation in Linked Lists

- Each node can be created as needed

```c
struct Node {
    int data;
    struct Node* next;
};
int main() {
    struct Node* head = NULL;
    struct Node* second = NULL;
    struct Node* third = NULL;
    // allocate 3 nodes in the heap
    head = (struct Node*)malloc(sizeof(struct Node));
    second = (struct Node*)malloc(sizeof(struct Node));
    third = (struct Node*)malloc(sizeof(struct Node));
    // populate the nodes with data
    head->data = 1;
    head->next = second;
    second->data = 2;
    second->next = third;
    third->data = 3;
    third->next = NULL;
}
```

Dynamic Memory Allocation in Structures

- Can be created dynamically as needed

```
struct Employee {
    int id;
    char name[50];
    float salary;
};
int main() {
    struct Employee* emp1 = (struct Employee*)
     malloc(sizeof(struct Employee));
    struct Employee* emp2 = (struct Employee*)
     malloc(sizeof(struct Employee));
    // populate the structure variables with data
    emp1->id = 1;
    strcpy(emp1->name, "John Doe");
    emp1->salary = 50000.00;
    emp2->id = 2;
    strcpy(emp2->name, "Jane Doe");
    emp2->salary = 60000.00;
}
```

Questions?