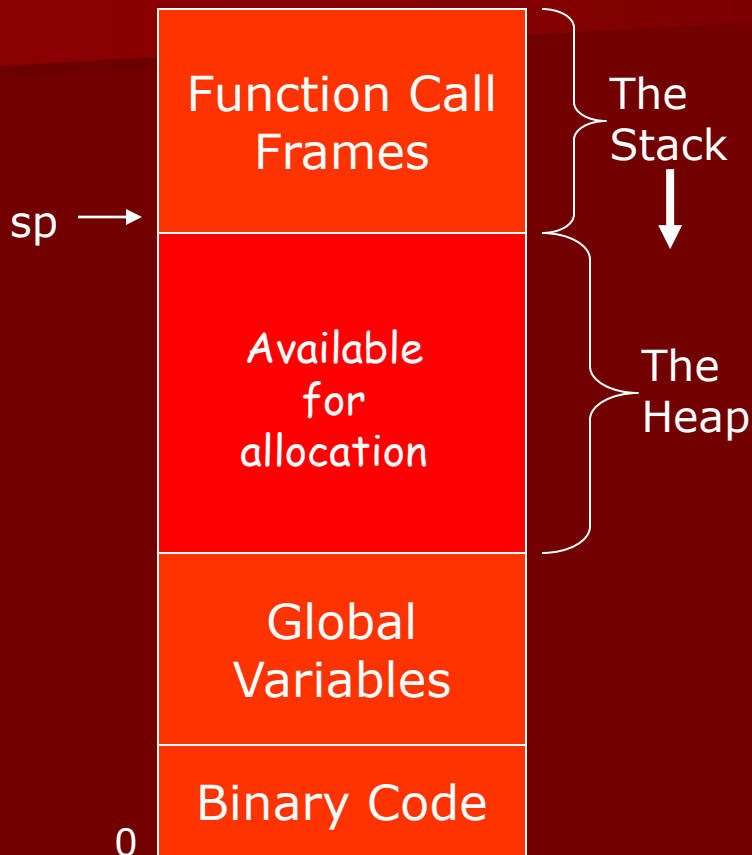


# Memory Management in C

Personal Software Engineering

# Memory Organization



- The call stack grows from the top of memory down
- Code is at the bottom of memory.
- Global data follows the code.
- What's left – the "heap" - is available for allocation.

# Allocating Memory

```
void *malloc( unsigned nbytes ) ;
```

- Allocates 'nbytes' of memory in the heap.
- Guaranteed not to overlap other allocated memory.
- Returns pointer to the first byte (or NULL if the heap is full).
- Similar to constructor in Java – allocates space.
- Space allocated uninitialized (random garbage).

```
void free( void *ptr ) ;
```

- Frees the memory assigned to ptr.
- The space must have been allocated by malloc.
- *No garbage collection in C (or C++).*
- Can slowly consume memory if not careful

# How Much Space Is Needed? - 1

`sizeof (type)` – gives the size of a type in bytes.

## Allocation Examples

```
int *ip ;
```



# How Much Space Is Needed? - 1


**sizeof** (*type*) – gives the size of a type in bytes.

## Allocation Examples

```
int *ip ;
```



```
ip = (int *) malloc( sizeof(int) ) ;
```



# How Much Space Is Needed? - 1

**sizeof** (*type*) – gives the size of a type in bytes.

## Allocation Examples

```
int *ip ;
```



A diagram showing a variable 'ip' (represented by a trapezoid) pointing to a rectangular box containing four question marks '????', representing an uninitialized memory block.

```
ip = (int *) malloc( sizeof(int) ) ;
```



A diagram showing a variable 'ip' (represented by a trapezoid) pointing to a rectangular box with a dot inside, representing a dynamically allocated memory block. An arrow points from this box to another rectangular box containing four question marks '????', indicating the memory has been allocated but not yet initialized.

```
*ip = 1234 ;
```



A diagram showing a variable 'ip' (represented by a trapezoid) pointing to a rectangular box with a dot inside, representing a dynamically allocated memory block. An arrow points from this box to another rectangular box containing the value '1234', indicating the memory has been allocated and initialized with the value 1234.

# How Much Space Is Needed? - 1

**sizeof** (*type*) – gives the size of a type in bytes.


## Allocation Examples

```
int *ip ;
```




A diagram showing a variable `ip` (represented by a trapezoid) pointing to a rectangular box containing four question marks `????`.

```
ip = (int *) malloc( sizeof(int) ) ;
```



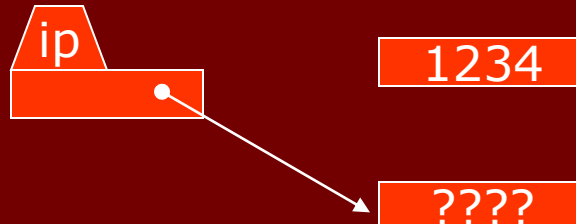
A diagram showing a variable `ip` (represented by a trapezoid) pointing to a rectangular box. Inside the box is a dot and an arrow pointing to another rectangular box containing four question marks `????`.

```
*ip = 1234 ;
```



A diagram showing a variable `ip` (represented by a trapezoid) pointing to a rectangular box. Inside the box is a dot and an arrow pointing to another rectangular box containing the value `1234`.

```
ip = (int *)malloc( sizeof(int) ) ;
```



A diagram showing a variable `ip` (represented by a trapezoid) pointing to a rectangular box. Inside the box is a dot and an arrow pointing to a rectangular box containing four question marks `????`. To the right of this, there is another rectangular box containing the value `1234`.

# How Much Space Is Needed? - 1

**sizeof** (*type*) – gives the size of a type in bytes.

## Allocation Examples

```
int *ip ;
```



A diagram showing a variable 'ip' (represented by a blue trapezoid) pointing to a rectangular memory block containing the text '????'.

```
ip = (int *) malloc( sizeof(int) ) ;
```



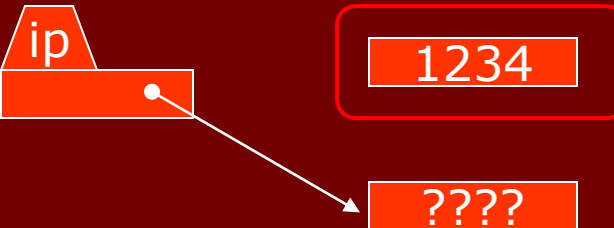
A diagram showing a variable 'ip' (represented by a blue trapezoid) pointing to a new rectangular memory block containing the text '????'. An arrow points from the new block to the previous block, which also contains '????'.

```
*ip = 1234 ;
```



A diagram showing a variable 'ip' (represented by a blue trapezoid) pointing to a rectangular memory block containing the text '1234'.

```
ip = (int *)malloc( sizeof(int) ) ;
```



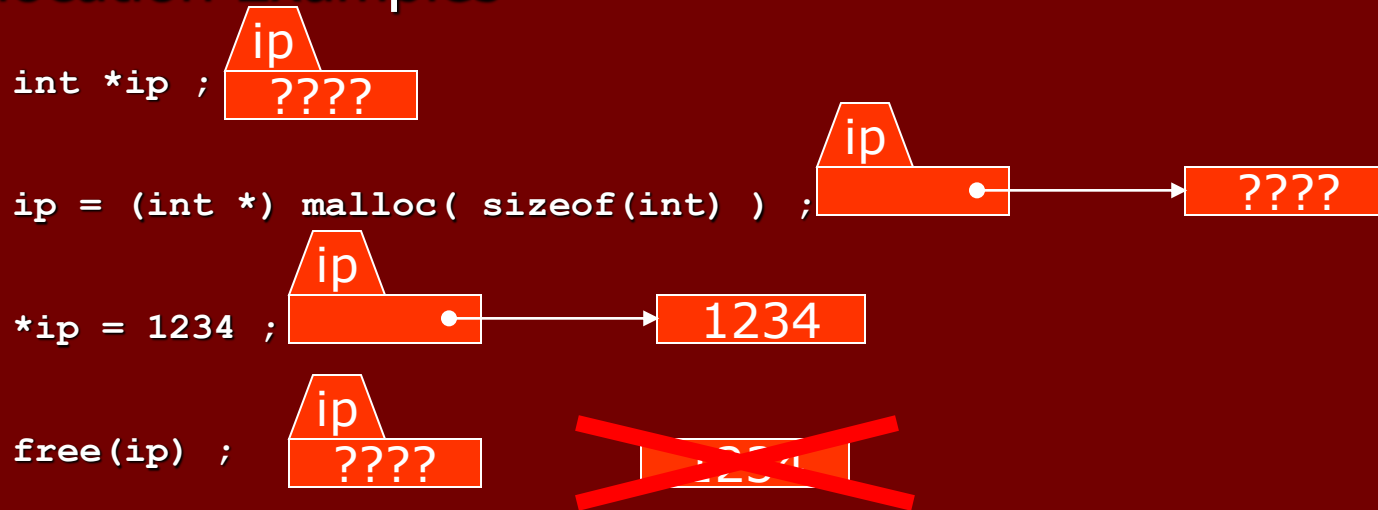
A diagram showing a variable 'ip' (represented by a blue trapezoid) pointing to a new rectangular memory block containing the text '????'. An arrow points from the new block to the previous block, which contains the text '1234'. A thought bubble labeled 'orphan storage' points to the '1234' block.



# How Much Space Is Needed? - 1

**sizeof** (*type*) – gives the size of a type in bytes.

## Allocation Examples



# How Much Space Is Needed? - 1

**sizeof** (*type*) – gives the size of a type in bytes.

## Allocation Examples

```
int *ip ;
```



The diagram shows a variable 'ip' (represented by a trapezoid) pointing to a rectangular box containing '????', representing an unallocated memory block.

```
ip = (int *) malloc( sizeof(int) ) ;
```



The diagram shows a variable 'ip' (represented by a trapezoid) pointing to a rectangular box containing a dot, which then points to another rectangular box containing '????', representing a newly allocated memory block.

```
*ip = 1234 ;
```



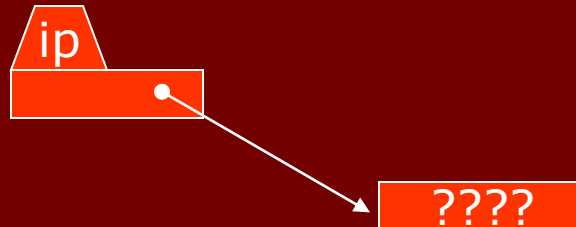
The diagram shows a variable 'ip' (represented by a trapezoid) pointing to a rectangular box containing a dot, which then points to another rectangular box containing the value '1234'.

```
free(ip) ;
```



The diagram shows a variable 'ip' (represented by a trapezoid) pointing to a rectangular box containing '????'. The box is crossed out with a large red 'X', indicating that the memory has been freed.

```
ip = (int *) malloc( sizeof(int) ) ;
```



The diagram shows a variable 'ip' (represented by a trapezoid) pointing to a rectangular box containing a dot, which then points to another rectangular box containing '????', representing a newly allocated memory block.

# How Much Space Is Needed? - 2

```
char *make_copy( char *orig ) {  
    char *copy = (char *) malloc( sizeof(char) * strlen(orig) + 1 ) ;  
    strcpy( copy, orig ) ;  
    return copy ;  
}
```

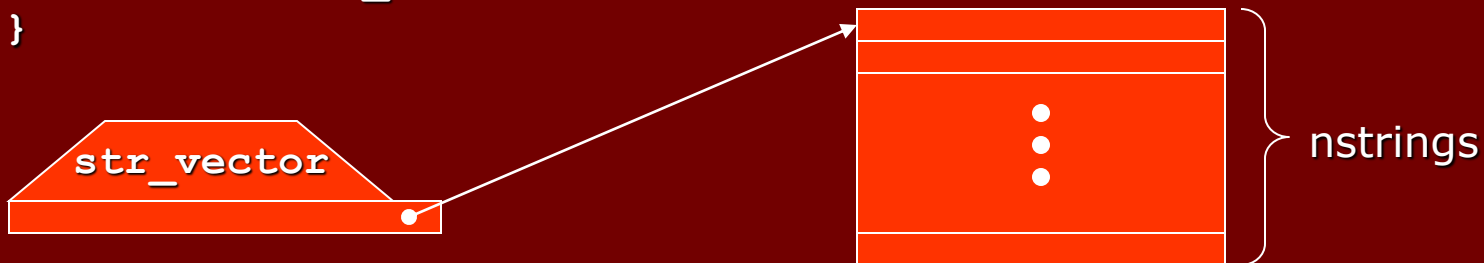
```
char orig[4] ;
```

'J'	'o'	'e'	'\0'
-----	-----	-----	------

```
char *copy
```



```
char **create_string_vector( int nstrings ) {  
    char **str_vector ;  
    str_vector = (char **) malloc( nstrings * sizeof(char *) ) ;  
    return str_vector ;  
}
```



# Linked Lists

- Structures with values and link (pointer) to next – and possibly previous - structure.
- Example: List of strings:

```
typedef struct node {  
    char *string ;  
    struct node *next ;  
} node ;  
  
node *top ;
```

