Structures and Dynamic Memory Allocation

NILIN PRABHAKER

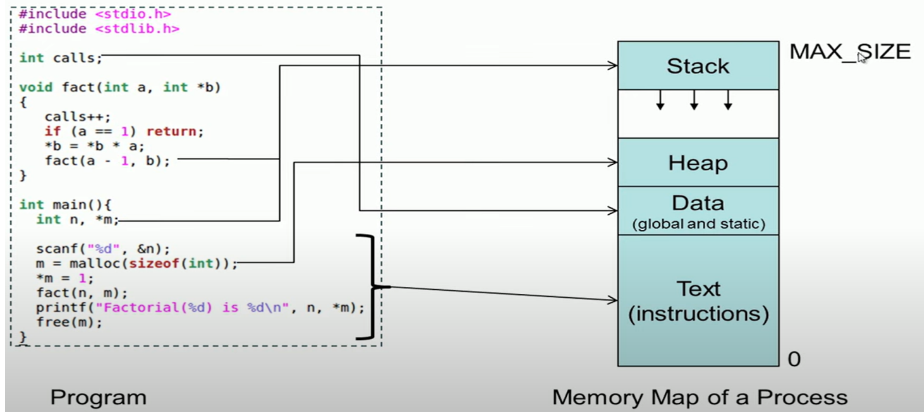
Neerukonda, Mangalagiri Mandal,
Guntur District, Andhra Pradesh
India, 522240

January 30, 2026

Outline

- Memory Map of a Process
- Structures and Pointer
- Dynamic Memory Allocation
- Linked Lists
    - Single
    - Doubly
    - Circular
- Trees
- Graphs
- Sorting and Searching Techniques

Memory Map of a Process



a

[a] Image Source: Information Security - 5 - Secure Systems Engineering, NPTEL, "Prof. Chester Rebeiro, IIT Madras"

SRM UNIVERSITY AP Andhra Pradesh

Structures

- Basic Data Types (int, float, char, etc.)
- Ordinary variables can hold one piece of information
- Arrays can hold a number of pieces of information of the same data type
- Entities that are collections of things
- Each thing having its own attributes
- Store data about a book

```c
int main( )
{
char name[3] ;
float price[3] ;
int pages[3], i ;
printf ( "\nEnter names, prices and no. of
    pages of 3 books\n" ) ;
for ( i = 0 ; i <= 2 ; i++ )
scanf ( "%c %f %d", &name[i], &price[i], &
    pages[i] );
printf ( "\nAnd this is what you entered\n" )
    ;
for ( i = 0 ; i <= 2 ; i++ )
printf ( "%c %f %d\n", name[i], price[i],
    pages[i] );
}
```

Structures

- Structure

```
int main( )
{
struct book
{char name ;
float price ;
int pages ;} ;
struct book b1, b2, b3 ;
printf ( "\nEnter names, prices & no. of
    pages of 2 books\n" ) ;
scanf ( " %c %f %d", &b1.name, &b1.price,
     &b1.pages ) ;
scanf ( " %c %f %d", &b2.name, &b2.price,
     &b2.pages ) ;
printf ( "\n%c %f %d", b1.name, b1.price,
     b1.pages ) ;
printf ( "\n%c %f %d", b2.name, b2.price,
     b2.pages ) ;}
```

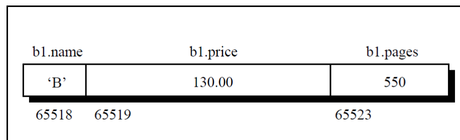
- Declaring a Structure

```
struct book
{
char name ;
float price ;
int pages ;
} b1, b2, b3 ;   OR

struct
{
char name ;
float price ;
int pages ;
} b1, b2, b3 ;
```

Structures

- Points to be Considered
  - Closing brace in the structure type declaration must be followed by a semicolon.
  - A structure type declaration does not tell the compiler to reserve any space in memory
  - Appears at the top of the source code file or a separate header file.
- Memory mapping



- Accessing Structure Elements

```
int main( )
{
struct book
{
char name ;
float price ;
int pages ;
} ;
struct book b1 = { 'B', 130.00,
    550 } ;
printf ( "\nAddress of price = %u"
    , &b1.price ) ;
printf ( "\nAddress of pages = %u"
    , &b1.pages ) ;
}
```

Structures

- Array of Structure

```
int main( ){
struct book
{    char name ;
     float price ;
     int pages ;
} ;
struct book b[100] ;
int i ;
for ( i = 0 ; i <= 99 ; i++ )
{    printf ( "\nEnter name, price and
    pages " ) ;
     scanf ( " %c %f %d", &b[i].name, &b[i
         ].price, &b[i].pages ) ;
}
for ( i = 0 ; i <= 99 ; i++ )
printf ( "\n%c %f %d", b[i].name, b[i].
    price, b[i].pages ) ;}
```

- Array of Structures
  - Each element of the array b is similar to the syntax used for arrays.
  - All elements of the array are stored in adjacent memory locations.

SRM UNIVERSITY AP Andhra Pradesh

Structures

- Additional Features of Structures
  - Values of a structure variable can be assigned to another structure variable of the same type using the assignment operator.
  - One structure can be nested within another structure.
  - A structure variable can also be passed to a function.
  - We can have a pointer pointing to a struct known as "structure pointers"
- Applications
  - Changing the size of the cursor
  - Clearing the contents of the screen
  - Placing the cursor at an appropriate position on screen
  - Drawing any graphics shape on the screen
  - Receiving a key from the keyboard
  - Checking the memory size of the computer
  - Hiding a file from the directory
  - Displaying the directory of a disk
  - Sending the output to printer
  - Interacting with the mouse etc.

SRM UNIVERSITY AP
Andhra Pradesh

Structures

- Assignment of Structure

```
void main( ){
struct employee
{    char name[10] ;
     int age ;
     float salary ;} ;
struct employee e1 = { "Sanjay",
    30, 5500.50 } ;
struct employee e2, e3 ;
strcpy ( e2.name, e1.name ) ;
e2.age = e1.age ;
e2.salary = e1.salary ; e3 = e2 ;
printf ( "\n%s %d %f", e1.name, e1
    .age, e1.salary ) ;
printf ( "\n%s %d %f", e2.name, e2
    .age, e2.salary ) ;
printf ( "\n%s %d %f", e3.name, e3
    .age, e3.salary ) ;}
```

- Nested Structures

```
int main( )
{
struct address
{    char phone[15] ;
char city[25] ;
int pin ;} ;
struct emp
{    char name[25] ;
struct address a ;} ;
struct emp e = { "jeru", "531046",
     "nagpur", 10 };
printf ( "\nname = %s phone = %s",
     e.name, e.a.phone ) ;
printf ( "\ncity = %s pin = %d", e
    .a.city, e.a.pin ) ;
}
```

Structures

- Structure variable as function parameter

```
int main( )
{
struct book
{ char name[25] ; char author[25]
    ;
     int callno ; } ;
struct book b1 = { "Let us C", "
    YPK", 101 } ;
display ( b1.name, b1.author, b1.
    callno ) ;}
display (char *s, char *t, int n )
{
printf ( "\n%s %s %d", s, t, n ) ;
}
```

- Pass Entire Structures

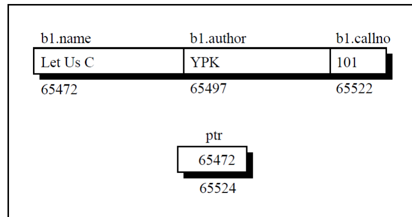
```
struct book
{    char name[25] ;
char author[25] ;
int callno ; } ;
int main( )
{
struct book b1 = { "Let us C", "
    YPK", 101 } ;
display ( b1 ) ;
}
display ( struct book b )
{
printf ( "\n%s %s %d", b.name, b.
    author, b.callno ) ;
}
```

Structures

■ Structure pointers

```
int main( ){
struct book
{    char name[25] ;
     char author[25] ;
     int callno ;} ;
struct book b1 = { "Let us C", "
    YPK", 101 } ;
struct book *ptr ;
ptr = &b1 ;
printf ( "\n%s %s %d", b1.name, b1
    .author, b1.callno ) ;
printf ( "\n%s %s %d", ptr->name,
    ptr->author, ptr->callno ) ;
}
```

■ Memory mapping



■ Self Referential Structure

```
struct Node {
int data; // Other data member
struct Node *next;
// Pointer to a structure of the
    same type
};
```

Dynamic Memory Allocation in C

- Allocate memory at runtime
- Allows us to handle data of varying size.
- The memory is allocated on the heap memory instead of the stack.
- The size of the memory can be increased if more elements are to be inserted and decreased of less elements are inserted.
- Allocated memory stays till function execution.
- Inbuilt-functions:
  - malloc()
  - alloc()
  - calloc()
  - free()

- malloc()

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

int main() {
int *ptr = (int *)malloc(20);

// Populate the array
for (int i = 0; i < 5; i++)
ptr[i] = i + 1;

// Print the array
for (int i = 0; i < 5; i++)
printf("%d ", ptr[i]);
return 0;
}
```

Dynamic Memory Allocation in C

- malloc()

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

int main() {
int *ptr = (int *)malloc(sizeof(
    int) * 5);
// Checking if failed or pass
if (ptr == NULL) {
    printf("Allocation Failed");
    exit(0);}
// Populate the array
for (int i = 0; i < 5; i++)
ptr[i] = i + 1;
// Print the array
for (int i = 0; i < 5; i++)
printf("%d ", ptr[i]);
return 0;}
```

- calloc()

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

int main() {
int *ptr = (int *)calloc(5, sizeof
    (int));
// Checking if failed or pass
if (ptr == NULL) {
    printf("Allocation Failed");
    exit(0);}
// No need to populate as already
// initialized to 0
// Print the array
for (int i = 0; i < 5; i++)
printf("%d ", ptr[i]);
return 0;}
```

Dynamic Memory Allocation in C

- free()

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

int main() {
int *ptr = (int *)calloc(5, sizeof
    (int));
// Do some operations.....
for (int i = 0; i < 5; i++)
printf("%d ", ptr[i]);
// Free the memory after
    completing
// operations
free(ptr);
ptr=NULL;
return 0;}
```

- realloc()

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

int main() {
int *ptr = (int *)malloc(5 *
    sizeof(int));
// Resize the memory block to hold
     10 integers
ptr = (int *)realloc(ptr, 10 *
    sizeof(int));
// Check for allocation failure
if (ptr == NULL) {
    printf("Memory Reallocation
        Failed");
    exit(0);}
return 0;}
```

Dangling Pointer

- Behavior of a pointer when its target has been deallocated or is no longer accessible.
- Types:
  - De-allocation of memory
  - Accessing an out-of-bounds memory location
  - When a variable goes out of scope
- Guidelines:
  - Always ensure that pointers are set to NULL after the memory is deallocated.
  - It will clearly signify that the pointer is no longer pointing to a valid memory location.
  - Avoid accessing a variable or a memory location that has gone out of scope.
  - Do not return pointers to local variables.

```c
#include <stdio.h>

int main(){

int *x = (int *)
    malloc(sizeof(int))
    ;
*x = 100;
printf("x: %d\n", *x);

free (x);
printf("x: %d\n", *x);
}
```

SRM UNIVERSITY AP Andhra Pradesh

Dangling Pointer

```c
#include <stdio.h>

int * function(); /*return Pointer*/

int main(){

int *x = function();
printf("x: %d", *x);
return 0;
}

int * function(){
int a =100;
return &a;
}
```

```c
#include <stdio.h>

int main(){
int *ptr;
int a = 10;{
    ptr = &a;
}

// 'a' is now out of scope
// ptr is a dangling pointer now
printf("%d", ptr);

return 0;
}
```

SRM UNIVERSITY AP Andhra Pradesh

Linked List

- Why linked list?
  - We are not sure about requirement of users.
  - Which user?
  - We are not sure about the exact size of array.
  - Example: Contact list, News Feeds, Messages, Number of Books in a library, etc.
  - Overflow (out of bound) and Underflow (wastage of memory).
  - Successive nodes of data objects were stored a fixed distance apart.
  - Insertion and Deletion of elements will be expensive.

SRM UNIVERSITY AP Andhra Pradesh

Linked List

- Benefits of Linked list
  - Item can be placed anywhere in the memory.
  - Efficient insertion and deletion.
  - Better memory utilization.
  - Ease of reordering.
  - Useful for implementing other Data Structures.
  - Ideal for Real-Time Applications.
- Limitation:

SRM UNIVERSITY AP
Andhra Pradesh

Thank You!