

Topics

- Structures, dynamic allocation review
- Memory Image based File operations
- Exercises

Dynamic Allocation

- Array variables have fixed size, used to store a fixed and known amount of variables – known at the time of compilation
- This size can't be changed after compilation
- However, we don't always know in advance how much space we would need for an array or a variable
- We would like to be able to dynamically allocate memory

The malloc function

void * malloc(unsigned int nbytes);

- The function malloc is used to dynamically allocate nBytes in memory
- malloc returns a pointer to the allocated area on success, NULL on failure
- You should always check whether memory was successfully allocated
- Remember to #include <stdlib.h>



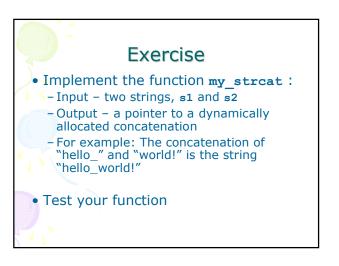
```
Example -dynamic_reverse_array
int main(void)
{
  int i, n, *p;
    printf("How many numbers do you want to enter?\n");
    scanf("%d", &n);

/* Allocate an int array of the proper size */
    p = (int *)malloc(n * sizeof(int));
    if (p = NULL)
    {
        printf("Memory allocation failed!\n");
        return 1;
    }
    /* Get the numbers from the user */
    ...
    /* Display them in reverse */
    /* Free the allocated space */
    free(p);
    return 0;
}
```

Why casting? The casting in p = (int *)malloc(n*sizeof(int)); is needed because malloc returns void *: void * malloc(unsigned int nbytes); The type (void *) specifies a general pointer, which can be cast to any pointer type.

Free the allocated memory void free (void *ptr); We use free (p) to free the allocated memory pointed to by p If p doesn't point to an area allocated by malloc, a run-time error occurs Always remember to free the allocated memory once you don't need it anymore





```
Solution: functionmy_strcat
char *my_strcat(char *str1, char *str2)
{
  int len1, len2;
  char *result;

  len1 = strlen(str1);
  len2 = strlen(str2);

  result = (char*)malloc((len1 + len2 + 1) *
  sizeof(char));
  if (result == NULL) {
    printf("Allocation failed! Check memory\n");
    return NULL;
}

strcpy(result, str1);
strcpy(result + len1, str2);
return result;
}
```

```
Solution: main()
int main(void)
{
    char str1[MAX_LEN + 1], str2[MAX_LEN + 1];
    char *cat_str;
    printf("Please enter two strings\n");
    scanf("%100s", str1);
    scanf("%100s", str2);
    cat_str = my_strcat(str1, str2);
    if (cat_str = NULL)
    {
        printf("Problem allocating memory!n");
        return 1;
    }

    printf("The concatenation of %s and %s is %s\n", str1, str2,
    cat_str);
    free(cat_str);
    return 0;
}
```



Structures - User Defined Types

- A collection of variables under a single name.
- A convenient way of grouping several pieces of related information together.
- Variables in a **struct** (short for structure) are called members or fields.

```
Defining a struct

struct struct-name
{

field-type1 field-name1;
field-type2 field-name2;
field-type3 field-name3;
...
};
```

Example – complex numbers

```
struct complex {
    int real;
    int img;
};
struct complex num1, num2,
    num3;
```

Typedef

• We can combine the typdef with the structure definition:

```
typedef struct complex {
    int real;
    int img;
} complex_t;

complex_t num1, num2;
```



```
Exercise

• Given two following structure:
typedef struct point
{
    double x;
    double y;
} point_t;

typedef struct circle
{
    point_t center;
    double radius;
} circle_t;

• Write a function is_in_circle which returns 1 if a point p est covered by circle c. Test this function by a program.
```

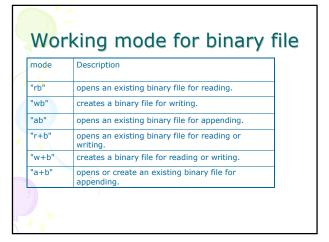


```
Solution
int is_in_circle(point_t *p, circle_t *c)
{
    double x_dist, y_dist;
    x_dist = p->x - c->center.x;
    y_dist = p->y - c->center.y;
    return (x_dist * x_dist + y_dist * y_dist <= c->radius * c->radius);
}
int main(void)
{
    point_t p;
    circle_t c;
    printf("Enter point coordinates\n");
    scanf("%lf%lf", &p.x, &p.y);
    printf("Enter circle center coordinates\n");
    scanf("%lf%lf", &c.center.x, &c.center.y);
    printf("Enter circle radius(n");
    scanf("%lf%lf", &c.radius);
    if (is_in_circle(fp, &c))
        printf("point is in circle\n");
    else
        printf("point is out of circle\n");
    return 0;
```

Pointers in Structures If a member in a struct is a pointer, all that gets copied is the pointer (the address) itself Exercise: Give this type of Student







File handle: Working with a bloc of data

- Two I/O functions: fread() and fwrite(), that can be used to perform block I/O operations.
- As other file handle function, they work with the file pointer.

fread()

The syntax for the fread() function is

size_t fread(void *ptr, size_t size,
 size_t n, FILE *stream);

- ptr is a pointer to an array in which the data is stored.
- size: size of each array element.
- n: number of elements to read.
- stream: file pointer that is associated with the opened file for reading.
- The fread() function returns the number of elements actually read.



fwrite()

The syntax for the fwrite() function is

```
size_t fwrite(const void *ptr, size_t
    size, size_t n, FILE *stream);
```

- ptr is a pointer to an array that contains the data to be written to an opened file
- n: number of elements to write.
- stream: file pointer that is associated with the opened file for writing.
- The fwrite() function returns the number of elements actually written.

function feof

- int feof(FILE *stream);
- return 0 if the end of the file has not been reached; otherwise, it returns a nonzero integer.

Examples

Read 80 bytes from a file.

```
enum {MAX_LEN = 80};
int num;
FILE *fptr2;
char filename2[]= "haiku.txt";
char buff[MAX_LEN + 1];
if ((fptr2 = fopen(filename2, "r")) == NULL) {
   printf("Cannot open %s.\n", filename2);
   reval = FAIL; exit(1);
}
. . . .
num = fread(buff, sizeof(char), MAX_LEN, fin);
buff[num * sizeof(char)] = `\0';
printf("%s", buff);
```

Exercise

- Write a program that use bloc-based file operations to copy the content of lab1.txt to to lab1a.txt
- Use: fread, fwrite, feof

