**DYNAMIC ALLOCATION AND DE-ALLOCATION OF MEMORY**

Memory for system defined variables and arrays are allocated at compilation time. The size of these variables cannot be varied during run time. These are called ‘*static data structures*’.

The *disadvantage* of these data structures is that they require fixed amount of storage. Once the storage is fixed if the program uses small memory out of it remaining locations are wasted. If we try to use more memory than declared overflow occurs.

If there is an unpredictable storage requirement, sequential allocation is not recommended. The process of allocating memory at run time is called *‘dynamic allocation’*. Here, the required amount of memory can be obtained from free memory called ‘*Heap*’, available for the user. This free memory is stored as a list called *‘Availability List’.* Getting a block of memory and returning it to the availability list, can be done by using functions like:

malloc() memory allocation

calloc() run time memory allocation

realloc() reallocation

free()

**FUNCTION MALLOC(SIZE)**

This function is defined in the header file <stdlib.h> and <alloc.h>. This function allocates a block of ‘*size’*  bytes from the heap or availability list. On success it returns a pointer of type ‘void’ to the allocated memory. We must typecast it to the type we require like int, float etc. If required space does not exist it returns NULL.

Syntax:

ptr = (data\_type\*) malloc(size);

where

ptr is a pointer variable of type data\_type.

data\_type can be any of the basic data type, user defined or derived data type.

size is the number of bytes required.

e.g.

ptr =(int\*)malloc(sizeof(int)\*n);

allocates memory depending on the value of variable n.

# include<stdio.h>

# include<string.h>

# include<alloc.h>

# include<process.h>

main()

{

char \*str;

if((str=(char\*)malloc(10))==NULL) /\* allocate memory for

string \*/

{

printf(“\n OUT OF MEMORY”);

exit(1); /\* terminate the program \*/

}

strcpy(str,”Hello”); /\* copy hello into str \*/

printf(“\n str= %s “,str); /\* display str \*/

free(str); /\* free memory \*/

}

In the above program if memory is allocated to the str, a string hello is copied into it. Then str is displayed. When it is no longer needed, the memory occupied by it is released back to the memory heap.

**FUNCTION CALLOC(N,SIZE)**

This function is defined in the header file <stdlib.h> and <alloc.h>. This function allocates memory from the heap or availability list. If required space does not exist for the new block or n, or size is zero it returns NULL.

Syntax:

ptr = (data\_type\*) calloc(n,size);

where

ptr is a pointer variable of type data\_type.

data\_type can be any of the basic data type, user defined or derived data type.

size is the number of bytes required.

n is the number of blocks to be allocated of size bytes.

and a pointer to the first byte of the allocated region is returned.

e.g.

# include<stdio.h>

# include<string.h>

# include<alloc.h>

# include<process.h>

main()

{

char \*str = NULL;

str=(char\*)calloc(10,sizeof(char)); /\* allocate memory for string \*/

if(str == NULL);

{

printf(“\n OUT OF MEMORY”);

exit(1); /\* terminate the program \*/

}

strcpy(str,”Hello”); /\* copy hello into str \*/

printf(“\n str= %s “,str); /\* display str \*/

free(str); /\* free memory \*/

}

**FUNCTION FREE(BLOCK)**

This function frees allocated block of memory using malloc() or calloc(). The programmer can use this function and de-allocate the memory that is not required any more by the variable. It does not return any value.

HEADER FILES

CTYPE.H

* “ctype.h” header file support all the below functions in C language.

|  |  |  |
| --- | --- | --- |
| **S.no** | **Function** | **Description** |
| 1 | [**isalpha()**](http://fresh2refresh.com/c/c-int-char-validation/c-isalpha-function/) | checks whether character is alphabetic |
| 2 | [**isdigit()**](http://fresh2refresh.com/c/c-int-char-validation/c-isdigit-function/) | checks whether character is digit |
| 3 | [**isalnum()**](http://fresh2refresh.com/c/c-int-char-validation/c-isalnum-function/) | checks whether character is alphanumeric |
| 4 | [**isspace()**](http://fresh2refresh.com/c/c-int-char-validation/c-isspace-function/) | checks whether character is space |
| 5 | [**islower()**](http://fresh2refresh.com/c/c-int-char-validation/c-islower-function/) | checks whether character is lower case |
| 6 | [**isupper()**](http://fresh2refresh.com/c/c-int-char-validation/c-isupper-function/) | checks whether character is upper case |
| 7 | [**isxdigit()**](http://fresh2refresh.com/c/c-int-char-validation/c-isxdigit-function/) | checks whether character is hexadecimal |
| 8 | [**isprint()**](http://fresh2refresh.com/c/c-int-char-validation/c-isprint-function/) | checks whether character is a printable character |
| 9 | [**ispunct()**](http://fresh2refresh.com/c/c-int-char-validation/c-ispunct-function/) | checks whether character is a punctuation |
| 10 | [**tolower()**](http://fresh2refresh.com/c/c-int-char-validation/c-tolower-function/) | checks whether character is alphabetic & converts to lower case |
| 11 | [**toupper()**](http://fresh2refresh.com/c/c-int-char-validation/c-toupper-function/) | checks whether character is alphabetic & converts to upper case |

MATH.H

* math.h” header file supports all the mathematical related functions in C language. All the arithmetic functions used in C language are given below. 5.67889
* round(a,2) ceil(4.5)=5

|  |  |
| --- | --- |
| **Function** | **Description** |
| [floor ( )](http://fresh2refresh.com/c/c-arithmetic-functions/c-floor-function/) | This function returns the nearest integer which is less than or equal to the argument passed to this function. |
| [round ( )](http://fresh2refresh.com/c/c-arithmetic-functions/c-round-function/) | This function returns the nearest integer value of the float/double/long double argument passed to this function. If decimal value is from ”.1 to .4″, it returns integer value less than the argument. If decimal value is from “.5 to .9″, it returns the integer value greater than the argument. |
| [ceil ( )](http://fresh2refresh.com/c/c-arithmetic-functions/c-ceil-function/) | This function returns nearest integer value which is greater than or equal to the argument passed to this function. |
| [sin ( 45)](http://fresh2refresh.com/c/c-arithmetic-functions/c-sin-cos-tan-exp-log-function/) | This function is used to calculate sine value. |
| [cos ( )](http://fresh2refresh.com/c/c-arithmetic-functions/c-sin-cos-tan-exp-log-function/) | This function is used to calculate cosine. |
| [log ( )](http://fresh2refresh.com/c/c-arithmetic-functions/c-sin-cos-tan-exp-log-function/) | This function is used to calculates natural logarithm. |
| [log10 ( )](http://fresh2refresh.com/c/c-arithmetic-functions/c-sin-cos-tan-exp-log-function/) | This function is used to calculates base 10 logarithm. |
| [sqrt ()](http://fresh2refresh.com/c/c-arithmetic-functions/c-sqrt-function/) | This function is used to find square root of the argument passed to this function. |
| [pow ( )](http://fresh2refresh.com/c/c-arithmetic-functions/c-pow-function/) | This is used to find the power of the given number. |

STRING.H

A string in the C language is simply an array of characters. Strings must have a NULL or \0 character after the last character to show where the string ends. A string can be declared as a character array or with a string pointer. First we take a look at a character array example:

char mystr[20];

As you can see the character array is declared in the same way as a normal array. This array can hold only 19 characters, because we must leave room for the NULL character.  
Take a look at this example:

#include<stdio.h>

int main()

{

char mystring[20];

mystring[0] = 'H';

mystring[1] = 'E';

mystring[2] = 'L';

mystring[3] = 'L';

mystring[4] = 'O';

mystring[5] = '\n';

mystring[6] = '\0';

printf("%s", mystring);

return 0;

}

**Note:** %s is used to print a string. (The 0 without the ” will in most cases also work).

String pointers are declared as a pointer to a char. When there is a value assigned to the string pointer the NULL is put at the end automatically. Take a look at this example:

#include<stdio.h>

int main()

{

char \*ptr\_mystring;

ptr\_mystring = "HELLO";

printf("%s\n", ptr\_mystring);

return 0;

}

It is not possible to read, with scanf, a string with a string pointer. You have to use a character array and a pointer. See this example:

#include<stdio.h>

int main()

{

char my\_array[10];

char \*ptr\_section2;

printf("Type hello and enter\n");

scanf("%s", my\_array);

ptr\_section2 = my\_array;

printf("%s\n", ptr\_section2);

return 0;

}

**string.h or strings.h**

The C language provides no explicit support for strings in the language itself. The string-handling functions are implemented in libraries. String I/O operations are implemented in <stdio.h> (puts , gets, etc). A set of simple string manipulation functions are implemented in <string.h>, or on some systems in <string**s**.h>.

clrscr();

getch();

getche();

The string library (string.h or strings.h) has some useful functions for working with strings, like strcpy, strcat, strcmp, strlen, strcoll, etc. We will take a look at some of these string operations.

**Important:** Don’t forget to include the library string.h (or on some systems strings.h) if you want to use one of these library functions.

**strcpy**

This library function is used to copy a string and can be used like this: strcpy(destination, source). (It is not possible in C to do this: string1 = string2). Take a look at the following example:

str\_one = "abc";

str\_two = "def";

strcpy(str\_one , str\_two); // str\_one becomes "def"

**Note:** strcpy() will not perform any boundary checking, and thus there is a risk of overrunning the strings.

**strcmp**

This library function is used to compare two strings and can be used like this: strcmp(str1, str2).

* If the first string is greater than the second string a number greater than null is returned.
* If the first string is less than the second string a number less than null is returned.
* If the first and the second string are equal a null is returned.

Take look at an example:

printf("Enter you name: ");

scanf("%s", name);

if( strcmp( name, "jane" ) == 0 )

printf("Hello, jane!\n");

**Note:** strcmp() will not perform any boundary checking, and thus there is a risk of overrunning the strings.

**strcat**

This library function concatenates a string onto the end of the other string. The result is returned. Take a look at the example:

printf("Enter you age: ");

scanf("%s", age);

result = strcat( age, " years old." ) == 0 )

printf("You are %s\n", result);

**Note:** strcat() will not perform any boundary checking, and thus there is a risk of overrunning the strings.

**strlen**

This library function returns the length of a string. (All characters before the null termination.) Take a look at the example:

name = "jane";

result = strlen(name); //Will return size of four.

#### ****Difference between structure and union in C:****

|  |  |  |
| --- | --- | --- |
| **S.no** | **C Structure** | **C Union** |
| 1 | Structure allocates storage space for all its members separately. | Union allocates one common storage space for all its members. Union finds that which of its member needs high storage space over other members and allocates that much space |
| 2 | Structure occupies higher memory space. | Union occupies lower memory space over structure. |
| 3 | We can access all members of structure at a time. | We can access only one member of union at a time. |
| 4 | Structure example: struct student { int mark; char name[6]; double average; }; | Union example: union student { int mark; char name[6]; double average; }; |
| 5 | For above structure, memory allocation will be like below. int mark – 2B char name[6] – 6B double average – 8B  Total memory allocation = 2+6+8 = 16 Bytes | For above union, only 8 bytes of memory will be allocated since double data type will occupy maximum space of memory over other data types.  Total memory allocation = 8 Bytes |