CPE102 Programming II

Week 6

Pointers

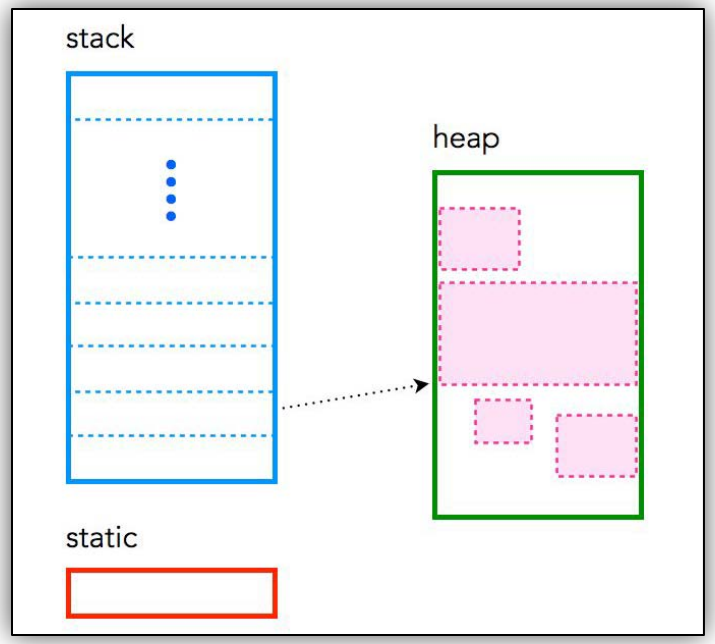
Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

Dynamic Memory Allocation

- When a program executes, the operating system reserves space to run program (stack and heap).
- The stack is a memory space where functions and their locally defined variables reside.
- The heap is reserved for program and it is an empty section to be used for allocating memory at runtime.

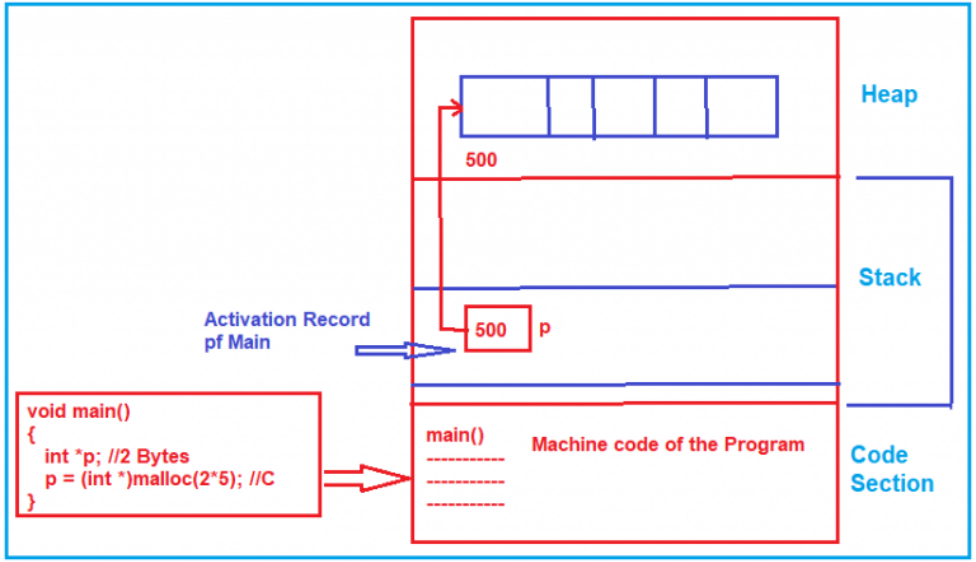
Stack and Heap

- Stack and heap are the logical parts of memory.
- Stack works in LIFO (Last in First Out) principal.(the stored information is stacked up)
- Stack: local variable storage (automatic, continuous memory, limited[If a program tries to put too much information on the stack, stack overflow will occur]).
- Heap: dynamic storage (large pool of memory, not allocated in contiguous order).
- Heap is a dynamic storage; you have to explicitly allocate (using malloc …), and deallocate (e.g. free) the memory.



The Heap

- Heap is slower than stack.
- the heap is managed by the programmer.
- in C, variables(at heap) are allocated and freed using functions like malloc() and free().
- the heap is large, and is usually limited by the physical memory available.
- the heap requires **pointers** to access it



Dynamic Memory Allocation

- An array is a collection of a fixed number of values. Once the size of an array is declared, you cannot change it.
- Sometimes the size of the array you declared may be insufficient. To solve this issue, you can allocate memory manually during run-time. This is known as dynamic memory allocation in C programming.
- To allocate memory dynamically, library functions are malloc(), calloc(), realloc() and free() are used.
- These functions are defined in the <stdlib.h> header file.

malloc() function

- The name "malloc" stands for memory allocation.
- The malloc() function reserves a block of memory of the specified number of bytes. And, it returns a pointer of void which can be casted into pointers of any form.

- Syntax of malloc()

  ptr = (castType*) malloc(size);
- Example

  ptr = (float*) malloc(100 * sizeof(float));

calloc() function

- The name "calloc" stands for contiguous allocation.
- The malloc() function allocates memory and leaves the memory uninitialized, whereas the calloc() function allocates memory and initializes all bits to zero.

- Syntax of calloc()

  ptr = (castType*)calloc(n, size);
- Example:

  ptr = (float*) calloc(25, sizeof(float));

- The above statement allocates contiguous space in memory for 25 elements of type float.

realloc() function

- If the dynamically allocated memory is insufficient or more than required, you can change the size of previously allocated memory using the realloc() function.

- Syntax of realloc()

  ptr = realloc(ptr, x);

- Here, ptr is reallocated with a new size x.

free() function

- Dynamically allocated memory created with either calloc() or malloc() doesn't get freed on their own. You must explicitly use free() to release the space.

- Syntax of free()

  ```
  free(ptr);
  ```

- This statement frees the space allocated in the memory pointed by ptr.

Dynamic Memory Allocation & Arrays

```c
#include <stdio.h>
#include <stdlib.h>
int main(){
    // This pointer will hold the base address of the block created
    int* ptr;
    int n, i;
    // Get the number of elements for the array
    printf("Enter number of elements:");
    scanf("%d", &n);
    // Dynamically allocate memory using malloc()
    ptr = (int*)malloc(n * sizeof(int));
    // Check if the memory has been successfully allocated by malloc or not
    if (ptr == NULL) {
        printf("Memory not allocated.\n");
        exit(0);}
    else {
        // Memory has been successfully allocated
        printf("Memory successfully allocated using malloc.\n");
        // Get the elements of the array
        for (i = 0; i < n; ++i)
            ptr[i] = i + 1;

        // Print the elements of the array
        printf("The elements of the array are: \n");
        for (i = 0; i < n; ++i)
            printf("Address:%d, Data: %10d \n", &ptr[i], ptr[i]);
    }

   free(ptr);
    return 0;
}
```

```
Enter number of elements:7
Memory successfully allocated using malloc.
The elements of the array are:
Address:12479168 ,     Data: 1
Address:12479172 ,     Data: 2
Address:12479176 ,     Data: 3
Address:12479180 ,     Data: 4
Address:12479184 ,     Data: 5
Address:12479188 ,     Data: 6
Address:12479192 ,     Data: 7
```

Function Pointers

- You learned that a pointer is a variable that holds the address of another variable. Function pointers are similar, except that instead of pointing to variables, they point to functions.
- A function name is really the starting address in memory of the code that performs the function's task.

```
int (*fPtr) (int,int)
```

In this definition, fPtr shows the address of a function(pointer to function) that takes two integer parameters and returns an integer value.

Function Pointers example

FunctionPtr.c

```c
#include <stdio.h>
void add(int a, int b){ printf("Addition is %d\n", a+b);}

void subtract(int a, int b){ printf("Subtraction is %d\n", a-b);}
void multiply(int a, int b){printf("Multiplication is %d\n", a*b); }

int main()
{
    // fun_ptr is a function pointers
    void (*fun_ptr)(int, int);
    unsigned int ch, a = 15, b = 10;

    printf("a = 15, b = 10 \n   Enter Choice: \n 0- add \n 1- subtract\n 2- multiply\n");
    scanf("%d", &ch);

    if (ch == 0)              fun_ptr= add;
    else if (ch == 1)     fun_ptr= subtract;
     else if (ch == 2)     fun_ptr= multiply;

     else
    {printf("Wrong choice\n");
        return 0;}
    fun_ptr(a,b);

    return 0;
}
```

```
a = 15, b = 10
   Enter Choice:
 0- add
 1- subtract
 2- multiply
2
Multiplication is 150
```

Example 1

- Write a program that continuously takes a character unless user press ENTER and prints “*” for each character entered from keyboard.
- When user presses ENTER the program will write all the characters entered since the beginning of data entrance in input order. Character code for “ENTER” is 13.

Example 1

- Without dynamic memory allocating it will be limited.

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

int main(){
    char giris[50];
    char *p;
    int i=0,k;
    p=giris;
    while(1){
        *(p+i)=getch();
        if(*(p+i)==13)
            break;
        putchar('*');
        printf("Adres[%d]: %d\n",i, p+i);
        i++;
    }
    printf("\n");
    for(k=0;k<i;k++){
        printf("Adres[%d]: %d\n",k, p+k);
        putchar(*(p+k));
    }
    getchar;
    return 0;
}
```

Example 1

```c
#include <stdio.h>
#include <stdlib.h>
void main()
{
    char *p;
    int i=0, k;

    p = (char *) malloc(sizeof(char));
    while(1)
    {
        *(p+i) = getch();
        if(*(p+i) == 13) break;
        putchar('*');
        i++;
        p = (char *) realloc(p, (i+1)*sizeof(char));
    }

    putchar('\n');
    for(k=0;k<i;k++)
        putchar(*(p+k));
}
```

Example 2

- Write a function with prototype given below which interchanges two variables values.

  void swap (int*, int*)

```
#include <stdio.h>
void swap(int * q,int * p)
{
  int temp = *p;
  *p = *q;
  *q = temp;
}

int main()
{
  int a = 10, b = 2, x = 3, y = 5;
  printf("a,b,x,y: %d,%d,%d,%d\n", a, b, x, y);
  swap(&x, &y);
  swap(&a, &b);
  printf("a,b,x,y: %d,%d,%d,%d\n", a, b, x, y);
}
```

Example 3

- Write a function with prototype given below which calculates the area and perimeter of a rectangle.

  void rectangle(int a,int b, int *area, int *perimeter)

```c
#include <stdio.h>
 void rectangle(int a, int b, int *area, int *perimeter);
int main() {
    int x, y;
    int area, perimeter;
    printf("Please enter the length and the width of the rectangle:\n " );
    scanf("%d %d", &x, &y);
    rectangle(x, y, &area, &perimeter);
     printf("the area= %d   , the perimeter =  %d\n", area, perimeter);
     }
void rectangle(int a,int b, int *area,int *perimeter) {
    *area = a * b;
    *perimeter = 2 * (a + b);
}
```

Example 4

- Write a function that performs like strlen function. Prototype for this function is as given below:

  int StrLen(char *)

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

int StrLen(char *);

int main() {
char str[100];
 printf("Enter string : ");
  gets(str);
  printf("Length of string : %d\n", StrLen(str));
   getch();
}
int StrLen(char * p) {
      int n =0;
   while(*p != '\0') {
     n++;
     p++;
       }
       return n;
}
```

Thanks ☺