# CPE102 Programming II

## Week 2

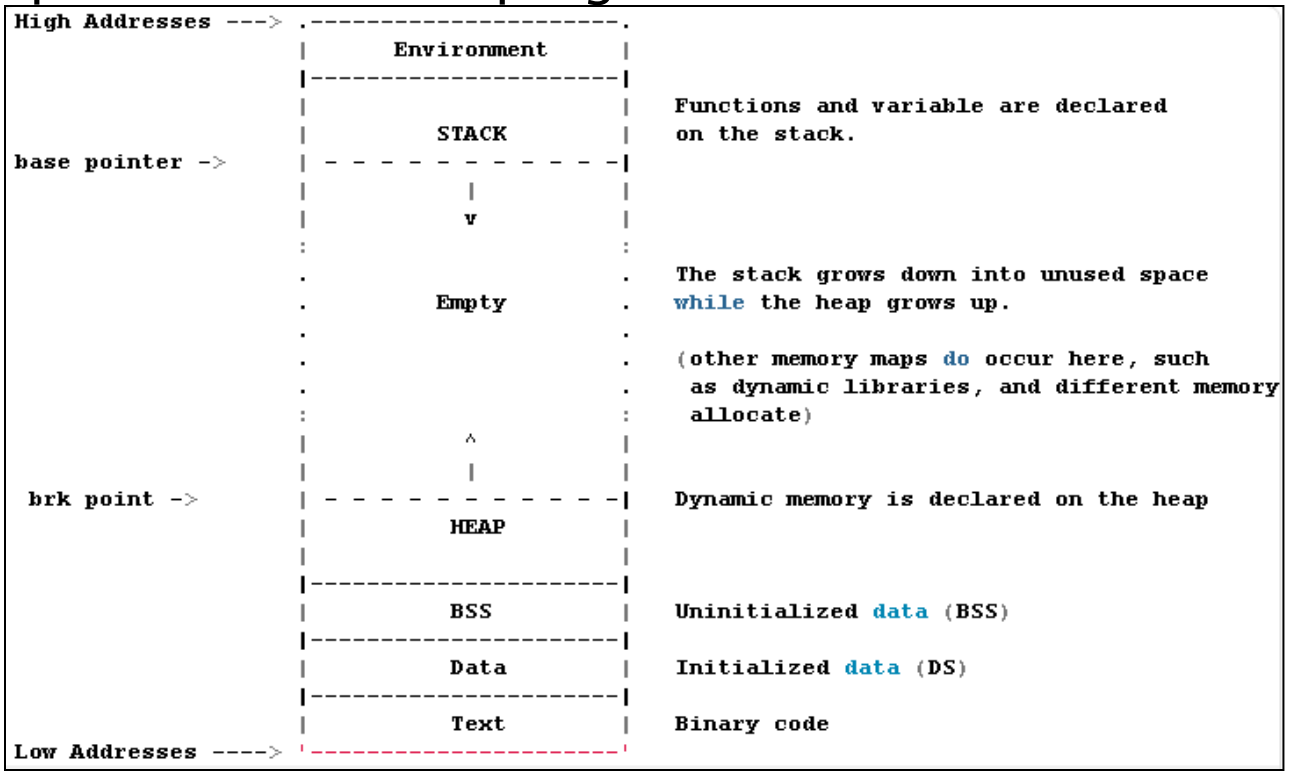
## *Memory Layout of C program, Generating Random Numbers, and Recursive Functions*

Dr. Nehad Ramaha,
Computer Engineering Department
Karabük Universities

# Memory Layout of C Programs

- A typical memory representation of C program consists of following sections:

1. Text segment
2. Initialized data segment
3. Uninitialized data Segment
4. Heap
5. Stack



# Memory Layout of C Programs

- Text segment: also known as a code segment or simply as text, is one of the sections of a program in an object file or in memory, which contains executable instructions.
- Usually, the text segment is sharable so that only a single copy needs to be in memory for frequently executed programs, such as text editors, the C compiler, the shells, and so on.
- Also, the text segment is often read-only, to prevent a program from accidentally modifying its instructions.

# Memory Layout of C Programs

- Initialized Data Segment (DS) : A data segment is a portion of virtual address space of a program (memory), which contains the global variables and static variables that are initialized by the programmer.

- Uninitialized Data Segment (BSS, block started by symbol): Data in this segment is initialized by the kernel to arithmetic '0' (zero) before the program starts executing uninitialized data starts at the end of the data segment and contains all global variables and static variables that are initialized to zero or do not have explicit initialization in source code.

# Memory Layout of C Programs

- Stack: where automatic variables are stored, along with information that is saved each time a function is called(local variables, parameters, and return values).
- It located at a higher address and grows and shrinks opposite to the heap segment.
- A stack frame will create in the stack when a function is called.
- Each function has one stack frame.
- Stack frames contain function's local variables, arguments and return value.
- This is how recursive functions in C can work.
- Each time a recursive function calls itself, a new stack frame is used, so one set of variables doesn't interfere with the variables from another instance of the function.
- Elements in a stack are added(push) or removed(pop) from the top of the stack, in a "last in , first out" or LIFO order.

# Memory Layout of C Programs

- Heap: It is used to allocate the memory at run time (dynamic memory allocation: a procedure in which the size of a data structure (like Array) is changed during the runtime).
- Heap area is managed by memory management functions like malloc, realloc, and free.

```c
#include <stdio.h>

int global; /* Uninitialized variable stored in bss*/
int main(void)
{
    int *ptr_one;
    ptr_one = (int *)malloc(sizeof(int)); /* memory allocating in heap segment */
    int c;//local variable stored in stack
    static int i = 100; /* Initialized static variable stored in DS*/
    static int k; /* Initialized static variable stored in bss*/
    return 0;
}
```

# Random Number Generator

- rand function
- <stdlib.h> library is needed
- Returns a “random” number between 0 and RAND_MAX ( at least 32767-max value for 16 bit integer)
- RAND_MAX is a symbolic constant defined in <stdlib.h>.
- Every number between 0 and RAND_MAX has equal probability of being chosen.
- The range of values produced by rand varies by what is needed in application.

# Random Number Generator


- Program simulating coin tossing might require only 1 for tails or 0 for heads.
- A dice rolling program that simulates six-sided die would requires random integers from 1 to 6.
- Scaling:
  - Values generated by rand is always between 0 and RAND_MAX
  - $0 \leq$ rand ( ) $\leq$ RAND_MAX
  - Use remainder % operator with rand function for example to produce numbers between 0 and 5. It is called scaling.
  - rand( ) % 6
  - The number 6 is called scaling factor

# Random Number Generator

- To shift the range just add 1 to the produced result.
- randNumber= 1 + rand ( ) % 6 produces numbers 1 ≤ randNumber≤ 6
- General rule: n = a + rand ( ) % b ;
  - a is shifting value (First number in desired range of consecutive integers)
  - b is scaling factor ( Equal to the width of desired range of consecutive integers)

# Scaled Random example (p206/ the book)

RandScaling.c

```c
//Fig. 5.11: fig05_11.c
//shifted, scaled random integers produced by 1 + rand() % 6.
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    unsigned int i;
    for( i = 1; i <= 20; ++i)
        {
            // pick random number from 1 to 6 and output it
            printf( "%10d", 1 + (rand() %6));
            // if counter is divisible by 5, begin new line of output
            if (i % 5 == 0)
                {
                    puts( "" );
                }
    }
}
```

Outputs:

```
     6     6     5     5     6
     5     1     1     5     3
     6     6     2     4     2
     6     2     3     4     1
```

# Random Number Generator Example:

- Sample Program: Simulating 6000 rolls of a die
- Each integer from 1 to 6 should appear approximately with equal likelihood (1000 times)

Rolling6000.c

```c
// Fig. 5.12: fig05_12.c
// Rolling a six-sided die 6000 times.
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   unsigned int frequency1 = 0; // rolled 1 counter
   unsigned int frequency2 = 0; // rolled 2 counter
   unsigned int frequency3 = 0; // rolled 3 counter
   unsigned int frequency4 = 0; // rolled 4 counter
   unsigned int frequency5 = 0; // rolled 5 counter
   unsigned int frequency6 = 0; // rolled 6 counter

   // loop 6000 times and summarize results
   for (unsigned int roll = 1; roll <= 6000; ++roll) {
   int face = 1 + rand() % 6; // random number from 1 to 6

   // determine face value and increment appropriate counter
   switch (face) {

   case 1: // rolled 1
   ++frequency1;
   break;

   case 2: // rolled 2
   ++frequency2;
   break;

   case 3: // rolled 3
   ++frequency3;
   break;

   case 4: // rolled 4
   ++frequency4;
   break;

   case 5: // rolled 5
   ++frequency5;
   break;

   case 6: // rolled 6
   ++frequency6;
   break; // optional
   }
   }
   // display results in tabular format
   printf(" \n", "Face", "Frequency");
   printf(" 1%13u\n", frequency1);
   printf(" 2%13u\n", frequency2);
   printf(" 3%13u\n", frequency3);
   printf(" 4%13u\n", frequency4);
   printf(" 5%13u\n", frequency5);
   printf(" 6%13u\n", frequency6);
   }
```

## Outputs:

```
Face      Frequency
   1           1003
   2           1017
   3            983
   4            994
   5           1004
   6            999
```

# Random Number Generator

- Function rand actually generates pseudorandom numbers
- Calling rand repeatedly produces a sequence of numbers that appears to be random.
- However, sequence repeats itself on each program execution
- To produce different sequence of integers on each program execution we use srand function
- srand takes an unsigned integer as an argument
- srand seeds rand function to produce different sequence of numbers on each execution of the program.

# Use srand function example (p209/ the book)

RandomizeSeed.c

```c
// Fig. 5.13: fig05_13.c
// Randomizing the die-rolling program.
#include <stdio.h>
#include <stdlib.h>


int main(void){
    unsigned int seed;
    // number used to seed the random number generator
    printf("%s", "Enter seed: ");
    scanf("%u", &seed);
    // note %u for unsigned int
    srand(seed);
    // seed the random number generator
    // loop 10 times
    for (unsigned int i = 1; i <= 10; ++i) {
            // pick a random number from 1 to 6 and output it
            printf("%10d", 1 + (rand() %6));
    // if counter is divisible by 5, begin a new line of output
        if (i % 5 == 0) {
            puts("");
        }
    }
}
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

```
Enter seed: 867
         2         4         6         1         6
         1         1         3         6         2
```

```
Enter seed: 67
         6         1         4         6         2
         1         6         1         6         4
```

# Random Number Generator

- To generate a random number each time without entering a seed value
- srand( time (NULL));
  - Reads system clock to obtain the value for seed automatically
  - Function time return the number of seconds that have passed since midnight on January 1970
  - The <time.h> library is used for the time function.

# Homework-1 (CRAPSGAME)

## Rules

- Roll two dice(hint: use rand)
- Sum of the spots on two upward faces is calculated
- 7 or 11 on first throw player wins
- 2, 3 or 12 on first throw player loses
- First throw of 4,5,6,8,9,10 becomes players point.
- Player must roll his dices till he win or lose:
  - If he rolled his point before rolling 7 he wins, if he rolled 7 he loses.

## Notes

- Upload your answer as c file to the first Assignment tag at https://oys.karabuk.edu.tr/
- Don't send homework to my email.
- You have from 10/3/2021 till 17/3/2021, no extension will given.
- Only one c file will be accepted.
- Don't use "enum", use only what we have learned till now + your previous knowledge from programming I.

*Player wins on the first roll*

```
Player rolled 5 + 6 = 11
Player wins
```

*Player wins on a subsequent roll*

```
Player rolled 4 + 1 = 5
Point is 5
Player rolled 6 + 2 = 8
Player rolled 2 + 1 = 3
Player rolled 3 + 2 = 5
Player wins
```

*Player loses on the first roll*

```
Player rolled 1 + 1 = 2
Player loses
```

*Player loses on a subsequent roll*

```
Player rolled 6 + 4 = 10
Point is 10
Player rolled 3 + 4 = 7
Player loses
```

# Creating Large Programs

- Typically, a large program is written in a separate directory as a collection of .h and .c file, with each .c file contains one or more functions definition.
- In particular, the #include directive tells the pre-processor to go read in the contents of a particular file, and place the contents inside the file being compiled at this point. The effect is as if you copied one file and pasted it into the other.
- There are two common formats for #includes, as follows:
  - #include < libraryFile.h > // The angle brackets say to look in the standard system directories
  - #include " personalHeaders.h" // The quotation marks say to look in the current directory.
- If it cannot be found, preprocessor issues an error message and compilations stops.
- Files with .h extension can include #include, #define directives, struct structures, function prototypes.

# Creating Large Programs

```c
#include "pgm.h"

int main(void)
{
        int i;
        for (i = 0; i < N;i++)
                f2();

        return 0;
}
```

program.c

```c
#include <stdio.h>
#define N 5

void f2(void)
{
        printf("Hello from f2()\n");
}
```

pgm.h

# Recursion

- A recursive function is one that calls itself.
- If a recursive function is called with a base case(simplest case) it returns the result.
- If a function is called with a more complex problem(such as 5!), the function divides the problem into two conceptual pieces:
  - First: a piece that function know how to do.
  - Second: a piece that function does not know how to do.
- The second part must resemble the original problem.
- The function launches a new copy of itself (recursion step) to solve what it does not know how to do.
- Eventually base case gets solved.

# Recursion example

- Program that prints numbers from 1 to N on the screen with recursion.

RecursionN.c

```
1  #include<stdio.h>
2
3  #define N 10
4
5  int PrintN(int n)
6      {
7       if(n== 0)
8          return 0;
9       PrintN(n-1);
10      printf("%d\n", n);
11 }
12
13 int main(void)
14 {
15     PrintN(N);
16     return 0;
17 }
18
```

# Recursion example

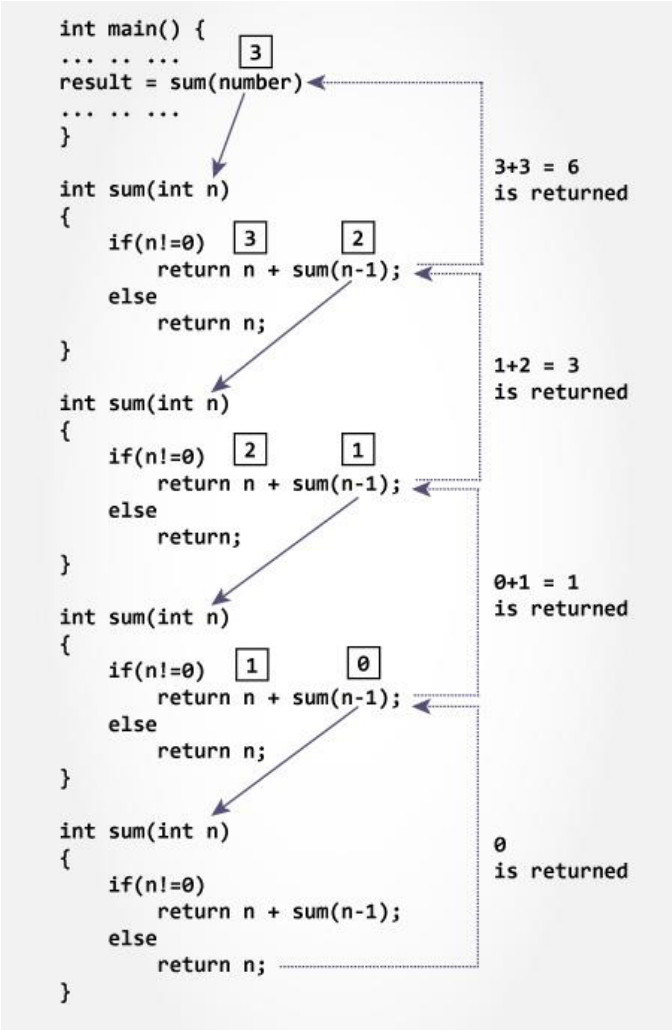
- A recursive function that finds the sum of the numbers from 1 to N.

RecursionSum.c

```c
#include<stdio.h>

int sum(int n)
   {
    if(n!= 1)
     return(n+ sum(n-1));
    else
     return n;
}

int main(void)
  {
   int N = 10;
   printf("sum = %d", sum(N));
   return 0;
}
```



Thanks ☺