
KOÇ
ÜNİVERSİTESİ

COMP304 PS 1

C Programming Basics

Najeeb Ahmad
nahmad16@ku.edu.tr

Outline

- C Program Structure
- Arrays
- Strings
- Structures
- Pointers
- Dynamic memory allocation/de-allocation
- Linked Lists
- Standard I/O
- Compiling and running C programs
  - Demo

The C language

- C Programming language
  - A powerful language suitable for both system programming (low level) and application programming
  - Flexible, efficient and portable to many platforms
  - Some applications of C
    - Operating systems development
    - Embedded systems
    - Compiler development
    - Databases
    - Device drivers
    - System programming
    - Computer simulations
    - Application software
    - Text editors
    - Computer Graphics
    - Game development
    - High Performance Computing

C Program Structure

```
// PREPROCESSOR DIRECTIVES

// GLOBAL DECLARATIONS


int main(void)
{
    // Local variables declarations

    // Statements

    return 0;
}


// FUNCTION DEFINITIONS
```

```
1  #include <stdio.h>
2  #define NUM_PROC 4
3
4  int num_threads = 8;
5  int addIntegers(int, int);
6
7  int main(void)
8  {
9    int num1 = 15, num2 = 34, sum = 0;
10   sum = addIntegers(num1, num2);
11
12   return 0;
13 }
14
15 int addIntegers(int a, int b)
16 {
17     return (a + b);
18 }
```

Arrays

- A collection of similar *type* of elements
  - Stored contiguously in memory
- These elements can be
  - of primitive types e.g. `int, double`
  - of derived types e.g. structures, pointers
- Advantages
  - Random access
  - Faster search for an element
- Disadvantages
  - Allocation size to be decided at compile time. May result in memory wastage
  - Insertion, deletion can be more costly

```
1  // Declaration by sizes
2  int pipes[10];
3  float velocities[20];
4
5  // Declaration by initialization
6  double velocity[]={1.4, 2.5, 6.7, 8.2};
7
8  // Declaration by size and initialization
9  int sz[10]={10, 20, 30, 40, 50};
10
11 int sum = sz[0]+sz[1]+sz[2]+sz[3];
12
13 // Multidimensional arrays
14 int matrix[10][10];
```

Arrays

```
1  #include <stdio.h>
2  #define SIZE 6
3
4  int main(void)
5  {
6      int m = 1.2, i = 0;
7      float c = -0.2;
8      float x[]={1.1, 2.4, 3.7, 4.8, 6.2, 7.5};
9      float y[SIZE];
10
11     for(i = 0; i < SIZE; i++)
12     {
13         y[i] = m * x[i] + c;       // y = 1.2x-0.2
14     }
15    return 0;
16 }
```

Strings

- A one-dimensional array of type ***char*** terminated by null character

```
1  // Declaration method 1
2  char course[]={'C','O','M','P','3','0','4','\0'};
3
4  // Declaration method 2
5  char course[]="COMP304";
```

- Some useful functions for string manipulation
  - `strlen(s1)`
    - Returns length of `s1` (excluding the null character)
  - `strcpy(s1, s2)`
    - Copy `s2` to `s1`
  - `strcmp(s1, s2)`
    - Compares `s1` and `s2`. Returns 0 if equal.
  - `strcat(s1, s2)`
    - Concatenate `s2` at the end of `s1`.

Strings

```
1  #include <stdio.h>
2  #include <string.h>
3
4  int main()
5  {
6      char course[]="COMP";
7      char code[]="304";
8      char favcourse[10];
9
10     int len1 = strlen(dept);            // len1 = 4
11     int len2 = strlen(course_code);     // len2 = 3
12
13     strcat(course, code);               // course="COMP304"
14     strcpy(favcourse, course);          // favcourse="COMP304"
15     return 0;
16 }
```

Structures

- Allow programmers to define their own data types
  - Useful for storing related information about an object
- Information about a person

```
char name[100];
char address[100];
int age;
long TC_number;
```

- Information stored as a *Person* structure

```
Struct Person
{
   char name[100];
   char address[100];
   int age;
   Long TC_number;
};
```

Structures

```c
#include <stdio.h>
#define PI 3.14159

struct Circle
{
    int center_x, center_y;
    int radius;
};

int main(void)
{
    // Initialization
    struct Circle c1 = {0, 0, 2};
    struct Circle c2 = {.center_x=1.2, .center_y=4, .radius=4};
    struct Circle c3;
    c3.center_x = 3.2;
    c3.center_y = 6.2;
    c3.radius = 4.1;
    double a1 = 0.0, a2 = 0.0;

    // Accessing structure fields
    a1 = PI * c1.radius * c1.radius;  // Area of c1
}
```

Structures

- Some structures in Linux
  - `task_struct`
    - Stores all the information about a linux process
    - Defined in `linux/sched.h` header file
  - `mem_map_t`
    - Stores information about physical memory pages
  - `mm_struct`
    - Describes virtual memory of a task or process
  - `inode`
    - Stores information about a file or directory on disk
  - `gendisk`
    - Stores information about a hard disk
  - `files_struct`
    - Stores information about open files in a process

Pointers

- Variables that store address of other variables or memory locations
  - Useful for dynamic memory allocation/de-allocation at runtime
  - Makes data exchange easier between program entities, in terms of time and memory consumption
  - Useful in construction of some data data structures, such as linked lists

```
1   int *intPtr;                // Asterisk before pointer variable name
2   float *floatPtr;
3   char *charPtr;
4
5   int ID=59486;
6   float perc=92.3;
7   char grade = 'A';
8
9   intPtr = &ID;              // Ampersand used to retrieve variable address
10  floatPtr = &perc;
11  charPtr = &grade;
12
13  // Asterisk to retrieve data from the address
14  printf("%d, %f, %c\n", *intPtr, *floatPtr, *charPtr);
```

Pointers

- Arithmetic operations on pointers

```
1   #include <stdio.h>
2   #define SIZE 4
3
4   int main(void)
5   {
6      int data[SIZE] = {10, 45, 90, 23};
7      int *dataPtr = &data[0];
8      // Print in original order
9      for(int i=0; i<SIZE; i++)
10     {
11         printf("%d\n", *dataPtr);
12         dataPtr++;
13     }
14     // Print reverse
15     dataPtr = &data[SIZE-1];
16     for(int i = SIZE; i > 0; i++)
17     {
18        printf(“%d\n”, *dataPtr);
19        dataPtr--;
20     }
21  }
```

- Array of pointers

```
1   #define SIZE 4
2
3   int main(void)
4   {
5       int quantity[] = {10, 30, 75, 40};
6       int *intPtr[SIZE];
7       int i = 0;
8       for(i=0; i < SIZE; i++)
9       {
10           intPtr[i] = &quantity[i];
11      }
12  }
```

- Pointer to Pointer
  - Pointer contains address of another pointer

```
1   int data=20;
2   int *ptr=&data;
3   int **ptrptr = &ptr;
4
5   printf("%d\n", **ptrptr);
```

Dynamic memory allocation

- It is possible to dynamically allocate/de-allocate memory for variables at runtime
  - Useful when exact amount of memory required is unknown at compile time
    - Overcomes limitation of arrays which require size to be known at compile time
  - Allows programs to manage memory at runtime
- C provides 4 library functions for dynamic memory management. Available through `stdlib.h`
  - `malloc`
  - `calloc`
  - `realloc`
  - `free`

Dynamic memory allocation

- `malloc`
  - Allocates a memory block with specified number of bytes

```
1   int main(void)
2   {
3       int n = 6, i = 0;
4
5       int *dataPtr = (int *)malloc(sizeof(int) * n);
6       if (dataPtr == NULL)
7       {
8           printf("Error allocating memory\n");
9           return -1;
10      }
11      for(int i = 0; i < n; i++)
12      {
13          dataPtr[i] = 2 * i;
14      }
15      for(int i = 0; i < n; i++)
16      {
17          printf("%d\n", dataPtr[i]);
18      }
19  }
```

- `calloc`
  - Allocates a memory block of the specified type, initializes to 0

```
int *dataPtr = (int *)calloc(n, sizeof(int));
```

- `realloc`
  - Reallocates/resizes a previous malloc, calloc allocation

```
dataPtr = (int *)realloc(dataPtr, (2n) * sizeof(int));
```

- `free`
  - De-allocates the allocated memory

```
free(dataPtr);
```

Linked Lists

- A collection of similar type of elements
  - Stored non-contiguously in memory
- These elements can be
  - of primitive types e.g. `int, double`
  - of derived types e.g. structures, pointers
- Advantages
  - Dynamic allocation
  - Easy insertion/deletion of elements
- Disadvantages
  - No random access
  - Extra memory space for pointers
  - Not cache-friendly

Linked Lists

```c
#include<stdlib.h>

struct Node
{
     int data;
     struct Node *nxt;
};

int main(void)
{
    struct Node* first = (struct Node*)malloc(sizeof(struct Node));
    struct Node* sec  = (struct Node*)malloc(sizeof(struct Node));
    struct Node* third = (struct Node*)malloc(sizeof(struct Node));

    first->data = 10;
    first->nxt = sec;
    sec->data = 20;
    sec->nxt = third;
    third->data = 30;
    third->nxt = NULL;
}
```

Linked Lists

```c
#include<stdlib.h>
#include<stdio.h>
#define SIZE 10;
struct Node
{
     int data;
     struct Node *nxt;
};

 void printList(struct Node* n)
{
      while(n!=NULL)
      {
          printf("%d\n", n->data);
          n = n->nxt;
      }
}
```

```c
int main(void)
{
   int i = 0;
   struct Node *nodes[SIZE];
   for(i = 0; i < SIZE; i++)
   {
      nodes[i] = (struct
Node*)malloc(sizeof(struct Node));
   }
   for(i = 0; i < SIZE-1; i++)
   {
      nodes[i]->data = 2 * i + 10;
      nodes[i]->nxt = nodes[i+1];
   }
   nodes[SIZE-1]->data =2 * (SIZE-1) + 10;
   nodes[SIZE-1]->nxt = NULL;
   printList(nodes[0]);
}
```

Linked Lists

- Linked List Types
  - Singly Linked List (Simple Linked List)
    - Each node contains data and pointer to next node
  - Doubly Linked List
    - Each node contains data and pointer to next and previous node

```
struct Node
{
    struct Node *prev;
    int data;
    struct Node *nxt;
};
```

  - Circular Linked List
    - Singly linked list in which last node points to the first node

Standard I/O

- Input
  - Reading data into a program
- Output
  - Writing data to a file, screen or printer etc.
- C treats all devices as files
- Standard input output files
  - stdin (input): Represents data input from the keyboard
  - stdout (output): Represents data output to the screen
- `scanf`: reads input from the stdin (keyboard)

```
char name[80];
Int age;
scanf("%s, %d", name, &age);
```

- `printf`: writes output to stdout (screen)

Standard I/O

- Input/Output to text files
  - Creating a text file and writing to it

```c
#include <stdio.h>

int main(void)
{
    int i = 0, in;
    FILE *fptr;
    fptr = fopen("myfile.txt","w")
    if(fptr == NULL)
    {
        printf("Could not create file\n");
        return -1;
    }
    for(i = 0; i < 10; i++)
    {
        scanf("%d", &in);
        fprintf(fptr, "%d\n", in);
    }
    fclose(fptr);
}
```

Standard I/O

- Input/Output to text files
  - Reading a text file

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       int i = 0, in;
6       FILE *fptr;
7       fptr = fopen("myfile.txt","r")
8       if(fptr == NULL)
9       {
10          printf("Could not open file for reading\n");
11          return -1;
12      }
13      for(i = 0; i < 10; i++)
14      {
15          fscanf(fptr, "%d", &in);
16          printf("%d\n", in);
17      }
18      fclose(fptr);
19  }
```

Standard I/O

- Input/Output to binary files
  - Normally takes much less memory than text files
  - File is not human readable when opened in a text editor
- Creating and writing to a binary file

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       int i = 0, in;
6       FILE *fptr;
7       fptr = fopen("myfile.bin","wb")
8       if(fptr == NULL)
9       {
10          printf("Could not open file for writing\n");
11          return -1;
12      }
13      for(i = 0; i < 10; i++)
14      {
15          scanf("%d", &in);
16          fwrite(&in, sizeof(int), 1, fptr);
17      }
18      fclose(fptr);
19  }
```

Standard I/O

- Reading from binary files

```
1   #include <stdio.h>
2
3   int main(void)
4   {
5       int i = 0, in;
6       FILE *fptr;
7       fptr = fopen("myfile.bin","rb")
8       if(fptr == NULL)
9       {
10          printf("Could not open file for reading\n");
11          return -1;
12      }
13      for(i = 0; i < 10; i++)
14      {
15          fread(&in, sizeof(int), 1, fptr);
16          printf("%d\n", in);
17      }
18      fclose(fptr);
19  }
```

Compiling and running program

- Write program in your favorite text editor
- Save it to a desired location as program_name.c, where program_name is your desired file name
- Open terminal in the location where you saved the file
  - Right click and select “Open in Terminal”
- Compiling the Program

```
gcc program_name.c -o myprogram
```

- Running the Program

```
./myprogram
```

- Compiling and running the program Demo
- For your assignments/projects, you many use C or C++

THANK YOU