CS1100 – Introduction to Programming

Instructor: Shweta Agrawal

Lecture 25

CS1100 – Introduction to Programming

Instructor: Shweta Agrawal

Lecture 25

- Data Types in C, Operators. Input and the Output.
- Modifying the control flow in Programs `if-else`, `switch`, loops : `while`, `do-while`, `for`.
- `Arrays` and `Strings` in C.
- Functions & modular programming.
- Recursion.

} So far...

CS1100 – Introduction to Programming

Instructor: Shweta Agrawal
Lecture 25

- Data Types in C, Operators. Input and the Output.
- Modifying the control flow in Programs `if-else`, `switch`, loops : `while`, `do-while`, `for`.
- `Arrays` and `Strings` in C.
- Functions & modular programming.
- Recursion.

} So far...

- Pointers in C, Pass by reference
- Dynamic memory allocation
- Structures in C

} Up Next...

Pointers – Recap

Pointers – Recap

- int count; – names a memory cell called count.
- Throughout the program the same memory cell gets accessed when we access count.
- The address of count is called its ℓ-value.
- The value of count (its *r*-value) may change during the course of the program.

Pointers – Recap

- int count; – names a memory cell called count.
- Throughout the program the same memory cell gets accessed when we access count.
- The address of count is called its ℓ-value.
- The value of count (its *r*-value) may change during the course of the program.

- int *countptr; – names a memory cell called countptr.

Pointers – Recap

- int count; – names a memory cell called count.
- Throughout the program the same memory cell gets accessed when we access count.
- The address of count is called its ℓ-value.
- The value of count (its *r*-value) may change during the course of the program.

- int *countptr; – names a memory cell called countptr.
- Throughout the program the same memory cell gets accessed when we access countptr as ℓ-value.
- However different cells may get accessed when we access countptr as *r*-value

Pointers – Recap

- int count; – names a memory cell called count.
- Throughout the program the same memory cell gets accessed when we access count.
- The address of count is called its ℓ-value.
- The value of count (its *r*-value) may change during the course of the program.

- int *countptr; – names a memory cell called countptr.
- Throughout the program the same memory cell gets accessed when we access countptr as ℓ-value.
- However different cells may get accessed when we access countptr as *r*-value which is the ℓ-value of some other variable.

An Application : Passing Parameters to Functions

A correct swap function :

```
#include<stdio.h>
void swap(int *p1, int *p2)
{
     int t;
     t = *p1;
     *p1 = *p2;
     *p2 = t;
}
int main()
{
    int a = 10, b = 20;
    swap(&a, &b);
    printf("%d %d",a,b);
}
```

An Application : Passing Parameters to Functions

A correct swap function :

```
#include<stdio.h>
void swap(int *p1, int *p2)
{
      int t;
      t = *p1;
      *p1 = *p2;
      *p2 = t;
}
int main()
{
     int a = 10, b = 20;
     swap(&a, &b);
     printf("%d %d",a,b);
}
```

Incorrect Version :

```
#include<stdio.h>
void swap(int *p1, int *p2)
{
      int *temp;
      temp = p1;
      p1 = p2;
      p2 = temp;
}

int main()
{
     int a = 10, b = 20;
     swap(&a, &b);
     printf("%d %d\n",a,b);
}
```

Array of pointers

Goal: We wish to store the names of three students in our class – "Sai", "Narasimhan", "Lakshmi" in some appropriate data-type.

Array of pointers

Goal: We wish to store the names of three students in our class – "Sai", "Narasimhan", "Lakshmi" in some appropriate data-type.

- What data-structure will you use?

Array of pointers

Goal: We wish to store the names of three students in our class – "Sai", "Narasimhan", "Lakshmi" in some appropriate data-type.

- What data-structure will you use?
  How about `char Names[3][11]`?

Array of pointers

Goal: We wish to store the names of three students in our class – "Sai", "Narasimhan", "Lakshmi" in some appropriate data-type.

- What data-structure will you use?
  How about `char Names[3][11]`?
- Use `char* Names[3]`

Array of pointers

Goal: We wish to store the names of three students in our class – "Sai", "Narasimhan", "Lakshmi" in some appropriate data-type.

- What data-structure will you use?
  How about `char Names[3][11]`?
- Use `char* Names[3]`
  - "Names" is an array of pointers to characters.

Array of pointers

Goal: We wish to store the names of three students in our class – "Sai", "Narasimhan", "Lakshmi" in some appropriate data-type.

- What data-structure will you use?
  How about `char Names[3][11]`?
- Use `char* Names[3]`
  - "Names" is an array of pointers to characters.

```
#include<stdio.h>
main() {
    char *Names[3]={"Sai", "Narasimhan", "Lakshmi"};
    int i;
    for (i=0; i<3; i++) {
       printf("%s\n",Names[i]);
    }
}
```

An array of pointers

Goal: Read the three names from standard input.

An array of pointers

Goal: Read the three names from standard input.

```
#include<stdio.h>
main() {
    char *Names[3];
    int i;

    for (i=0; i<3; i++) {
        printf("Enter Name %d\t", i+1);
        scanf("%s", Names[i]);
    }
}
```

An array of pointers

Goal: Read the three names from standard input.

```
#include<stdio.h>
main() {
    char *Names[3];
    int i;

    for (i=0; i<3; i++) {
        printf("Enter Name %d\t", i+1);
        scanf("%s", Names[i]);
    }
}
```

This program is incorrect! There is no memory allocated for `Names[i]`. The program most likely gives a core dump.

An array of pointers – Another program

Goal: Read the three names from standard input.

An array of pointers – Another program

Goal: Read the three names from standard input.

```
#include<stdio.h>
int main() {
    char *Names[3]; char temp[100]; int i;

    for (i=0; i<3; i++) {
        scanf("%s", temp);
        Names[i] = temp;
        printf("String input %s\n",Names[i]);
    }
    for (i=0; i<3; i++) {
        printf("String output %s\n",Names[i]);
    }
}
```

An array of pointers – Another program

Goal: Read the three names from standard input.

```
#include<stdio.h>
int main() {
    char *Names[3]; char temp[100]; int i;

    for (i=0; i<3; i++) {
        scanf("%s", temp);
        Names[i] = temp;
        printf("String input %s\n",Names[i]);
    }
    for (i=0; i<3; i++) {
        printf("String output %s\n",Names[i]);
    }
}
```

This program is still incorrect! All 3 array locations point to the same array temp.

Allocating memory using malloc

- malloc – memory allocator – is a function that allocates memory to the program and returns a pointer to that memory.

Allocating memory using malloc

- malloc – memory allocator – is a function that allocates memory to the program and returns a pointer to that memory.
- ```
  int *ptr;
  ptr = (int *) malloc(sizeof(int));
  ```
  ```

Allocating memory using malloc

- malloc – memory allocator – is a function that allocates memory to the program and returns a pointer to that memory.
- ```
  int *ptr;
  ptr = (int *) malloc(sizeof(int));
  ```
- The input to malloc is size of the memory required.
- malloc returns a pointer to the memory allocated – the type of the pointer is `(void *)`.
  ```

Allocating memory using malloc

- malloc – memory allocator – is a function that allocates memory to the program and returns a pointer to that memory.
- ```
  int *ptr;
  ptr = (int *) malloc(sizeof(int));
  ```
- The input to malloc is size of the memory required.
- malloc returns a pointer to the memory allocated – the type of the pointer is `(void *)`.
- Note the typecasting into `(int *)`.
  ```

Allocating memory using malloc

- malloc – memory allocator – is a function that allocates memory to the program and returns a pointer to that memory.
- ```
  int *ptr;
  ptr = (int *) malloc(sizeof(int));
  ```
- The input to malloc is size of the memory required.
- malloc returns a pointer to the memory allocated – the type of the pointer is `(void *)`.
- Note the typecasting into `(int *)`.
- Memory obtained using malloc is destroyed only when it is explicitly freed or the program terminates.
- This is unlike variables which are unavailable outside their scope.
  ```

An array of pointers – a correct program

Goal: Read the three names from standard input.

An array of pointers – a correct program

Goal: Read the three names from standard input.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main() {
    char *Names[3]; char temp[100]; int i;
    for (i=0; i<3; i++) {
      scanf("%s", temp);
      Names[i]=(char *)malloc(sizeof(strlen(temp)));
      strcpy(Names[i], temp);
      printf("String input %s\n",Names[i]);
    }
    for (i=0; i<3; i++)
      printf("String output %s\n",Names[i]);
    return 0;
}
```

An array of pointers – a correct program

Goal: Read the three names from standard input.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
int main() {
    char *Names[3]; char temp[100]; int i;
    for (i=0; i<3; i++) {
      scanf("%s", temp);
      Names[i]=(char *)malloc(sizeof(strlen(temp)));
      strcpy(Names[i], temp);
      printf("String input %s\n",Names[i]);
    }
    for (i=0; i<3; i++)
      printf("String output %s\n",Names[i]);
    return 0;
}
```

Note the use of malloc and also the stdlib.h

2D Arrays using pointers

Consider the following declaration:
int nums[2][3] $= \{\{16, 18, 20\}, \{25, 26, 27\}\}$;
How to reference these elements using pointers?

2D Arrays using pointers

Consider the following declaration:
int nums[2][3] = $\{\{16, 18, 20\}, \{25, 26, 27\}\}$;
How to reference these elements using pointers?

In general, nums[i][j] is equivalent to *(*(nums+i)+j)

Pointer Notation	Array Notation	Value
*(*nums)	nums[0] [0]	16
*(*nums+1)	nums[0] [1]	18
*(*nums+2)	nums[0] [2]	20
((nums + 1))	nums[1] [0]	25
((nums + 1)+1)	nums[1] [1]	26
((nums + 1)+2)	nums[1] [2]	27

2D Arrays using pointers

Consider the following declaration:
int nums[2][3] = $\{\{16, 18, 20\}, \{25, 26, 27\}\}$;
How to reference these elements using pointers?

In general, nums[i][j] is equivalent to *(*(nums+i)+j)

Pointer Notation	Array Notation	Value
*(*nums)	nums[0] [0]	16
*(*nums+1)	nums[0] [1]	18
*(*nums+2)	nums[0] [2]	20
((nums + 1))	nums[1] [0]	25
((nums + 1)+1)	nums[1] [1]	26
((nums + 1)+2)	nums[1] [2]	27

Some more practice

- Consider the following declaration:
  char * ptr = “geek”;

Some more practice

- Consider the following declaration:
  char * ptr = “geek”;
- What is char x = *(ptr+3); ?

Some more practice

- Consider the following declaration:
  char * ptr = “geek”;
- What is char x = *(ptr+3); ?
- Null Pointer: We can create a null pointer by assigning null value during the pointer declaration.

Some more practice

- Consider the following declaration:
  char * ptr = “geek”;
- What is char x = *(ptr+3); ?
- Null Pointer: We can create a null pointer by assigning null value during the pointer declaration.
- This method is useful when you do not have any address assigned to the pointer.

Some more practice

- Consider the following declaration:
  char * ptr = “geek”;
- What is char x = *(ptr+3); ?
- Null Pointer: We can create a null pointer by assigning null value during the pointer declaration.
- This method is useful when you do not have any address assigned to the pointer.
- Declaration: int *p = NULL

Some more practice

- Consider the following declaration:
  char * ptr = “geek”;
- What is char x = *(ptr+3); ?
- Null Pointer: We can create a null pointer by assigning null value during the pointer declaration.
- This method is useful when you do not have any address assigned to the pointer.
- Declaration: int *p = NULL
- if(ptr) : succeeds if p is not null

Some more practice

- Consider the following declaration:
  char * ptr = “geek”;
- What is char x = *(ptr+3); ?
- Null Pointer: We can create a null pointer by assigning null value during the pointer declaration.
- This method is useful when you do not have any address assigned to the pointer.
- Declaration: int *p = NULL
- if(ptr) : succeeds if p is not null
- if(!ptr) : succeeds if p is null

More practice: Pointers and strings

```
#include <stdio.h>
#include <string.h>
int main()
{
char str[]="Hello Guru99!";
char *p;
p=str;
printf("First character is:%c\n",*p);
p =p+1;
printf("Next character is:%c\n",*p);
printf("Printing all the characters in a string\n");
p=str;  //reset the pointer
for(int i=0;i<strlen(str);i++)
{
printf("%c\n",*p);
p++;
}
return 0;
}
```