# CSEN403: Concepts of Programming Languages Imperative Programming II: C

Prof. Dr. Slim Abdennadher Dr. Nada Sharaf

Spring Semester 2021

## Compiling and Running a C Program

- Preprocessor
  - ► Handling comments, constant values, ... etc
- Compiler & Assembler
  - ▶ Produces a file with assembly code
  - Assembly Code is then translated to machine code
- Linker
  - Augments the missing parts and function calls and produces the executable file from the obj file

#### Scanning

```
scanf("%i", &counter);
scanf("%c", &Response);
scanf("%f", &_averageMileage);
scanf("%s", &line);
Note that & stands for the reference (address) of the variable.
```

#### Conditional Statements: If-then-else

```
if(<condition>){
    <statement(s)_block>
}
[else if(<condition>){
    <statements_block>
else{
    <statements_block>
}]
```

#### Conditional Statements: Switch Statements

```
switch(<var_name>) {
    case <cond1>: <statements_block>
        break;
    case <cond2>: <statements_block>
    .
    .
    default: <statements_block>
}
```

#### Iterative Statements: While

#### Iterative Statements: Do While

#### Iterative Statements: For Loop

## Tracing I

```
int a;
int *y;
int *x = &a;
*x = 15;
*x += 2;
y = x;
*y *= 2;
```

## Tracing II

```
#include < stdio . h>
int main()
     int a = 10, b = 10;
     int c.d:
     int *ptra = \&a;
     int *ptrb = \&b;
     ++*ptra;
     (*ptrb)++;
     printf("\n_1A_1=1\%d_1,11B_1=1\%d", a , b);
     c = ++*ptra;
     d = (*ptrb)++;
     printf("\n_{1}A_{1}=1\%d_{1}, \dots B_{n}=1\%d", a , b);
     printf("\n_\C_\=\\%d\,,\\D_\=\\%d\,, c , d);
     return 0:
```

## Tracing III

```
int name[] = {5,23,119};
int *p, *q;
p = name;
q = name + 1;
printf("%i_%i_%i_\n", *name, *p, *q);
*(p++);
(*q)++;
printf("%i_%i\n", *p, q[0]);
```

#### **Pointers**

Sending a copy vs. sending the address.

```
void swap(int x, int y){
int tmp;
tmp = x;
x = y;
y = tmp;
int main(){
int a = 2, b = 3;
swap(a, b);
return 0:
```

```
void swap(int *x, int *y){
int tmp;
tmp = *x;
*x = *y;
*y = tmp;
int main(){
int a = 2, b = 3;
swap(&a, &b);
return 0:
```

## Will this Swap?

```
void swap(int x, int y){
    int tmp;
    tmp = x;
    x = y;
    y = tmp;
int main(){
    int a = 2, b = 3;
    int *p = \&a;
    int *q = \&b;
    swap(*p, *q);
    return 0:
```

#### Pointer Summary

- If v is a variable then &v is the location/address in memory holding its value.
- Addresses are values which can be manipulated.
- Pointers are typed: a pointer to a char is different from a pointer to an int.
- Pointer Assignment: ptr = &i
- Pointer Dereferencing: i = \*ptr
- int \*ptr = NULL; /\* ptr = 0, points nowhere \*/

#### Structures In C

- Collection of variables
- Each variable can have a different type.
- A structure is a convenient way of grouping several pieces of related information together.
- A structure can be defined as a new named type, thus extending the number of available types.

#### Defining a Structure I

```
struct card {
  int value;
  char suit;
  };
Declared: struct card c1, c2;
```

## Defining a Structure II

```
typedef struct {
  char name[64];
  char course[128];
  int age;
  int year;
} student;
Declared: student st_rec;
```

#### Accessing variables

```
c1.value = 1;
c2.suit = 's'; /* c1.suit has type char */
st_rec.age = 23;
st_rec.name = {'S','L','I','M'};
```

#### Dynamic Memory Allocation

- Used to allocate memory dynamically at runtime
- If you do not know the needed size at compile time
- You must free the allocated memory afterwards using the funcion free.
- malloc
- malloc return a pointer that needs type casting.
- You can use the function sizeof
- printf("int needs %d bytes \n", sizeof(int));

#### Examples I

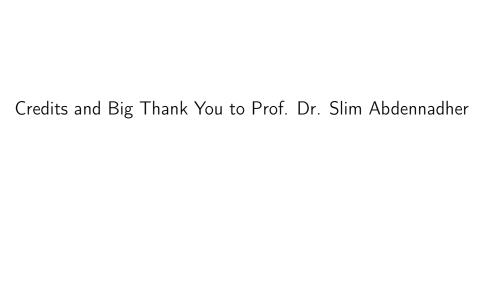
```
• int* arr = (int*)malloc(n);
```

#### Examples II

```
TreeNode* CreateTreeNode
                               (int data)
typedef struct TreeNode
                             TreeNode* ptr =
                             (TreeNode *)
 int data:
                             malloc(sizeof(TreeNode));
 struct TreeNode* left:
 struct TreeNode* right;
                             (*ptr).data = data;
                             (*ptr).left = NULL;
}TreeNode;
                             (*ptr).right = NULL;
                             return ptr;
```

## Examples III

```
void DeleteTreeNode(TreeNode* ptr)
{
    if( ptr == NULL) return;
    DeleteTreeNode( (*ptr).right);
    DeleteTreeNode( (*ptr).left);
    free( ptr );
}
```



## Thank you