NOVEMBER 25TH 2020

ELEMENTARY PROGRAMMING

SOME COVID BEST PRACTICES BEFORE WE START

- If you fill ill, go home
- Keep your distance to others
- Wash or sanitise your hands
- Disinfect table and chair
- Respect guidelines and restrictions

REMEMBER TO BOOK YOUR SPOT TO DISCUSS THE SECOND ASSIGNMENT

- If you didn’t receive my email please tell me I will resend the links
- You need to book an appointment otherwise no evaluation
  - Emanuele: **https://calendly.com/dierre/10min**
  - Alland: **https://calendly.com/a-kareem1991/02318_evaluering_1**
  - Patrick: **https://calendly.com/02318_opgave_eval_ph/10-min-eval**
  - Freja: **https://calendly.com/s200544/10-mins-samtale-om-aflevering**

NEW FEEDBACK

- I would really like for you to take a survey at the end of the session
- Feedback is important, please take the time to do it
- Pretty please <3
- Type this in your browser http://bit.ly/elemprog12

PASSING ARGUMENTS TO A C PROGRAM

```c
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf("Number of parameters is %d\n", argc);
    printf("The program name is %s\n", argv[0]);
    if (argc == 2) {
        printf("The argument supplied is %s\n", argv[1]);
    } else if (argc > 2) {
        printf("Too many arguments supplied, I only accept one.\n");
    } else {
        printf("Error: one argument expected.\n”);
    }
    printf("\nThis is all that was passed:\n”);
    for(int i = 0; i < argc; i++) {
      printf(" - %s\n", argv[i]);
    }
}
```

FUNCTIONS

- A function is a group of statements that together perform a task
- You can divide up your code into separate functions
- Every C program has at least one function, which is `main()`

FUNCTIONS

- A function declaration tells the compiler about a function's name, return type, and parameters
- A function definition provides the actual body of the function
- The C standard library provides numerous built-in functions that your program can call

DEFINING A FUNCTION

```
return_type function_name( type in1, …, type inN ) {
    body of the function
}
```

- `return_type` is the type of the data returned by the function. If you don't want to return data, you can use `void`
- function_name is the identifier of your function. You will use this identifier when you want to call

DEFINING A FUNCTION

```
return_type function_name( type in1, …, type inN ) {
    body of the function
}
```

- type in1, …, type inN are the arguments of your function
- body of the function is the actual content of the function, it's the collection of statements that will be executed

SIMPLE EXAMPLE OF A FUNCTION

```
int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
```

SIMPLE EXAMPLE OF A FUNCTION

declaration

definition

```c
#include <stdio.h>
int max(int num1, int num2);
int main() {
    int a = 100;
    int b = 200;
    int ret;
    ret = max(a, b);
    printf("Max value is : %d\n", ret);
    return 0;
}
int max(int num1, int num2) {
    int result;
    if (num1 > num2) {
        result = num1;
    } else {
        result = num2;
    }
    return result;
}
```

PASSING PARAMETERS TO A FUNCTION

- pass by value: this method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
- pass by reference: This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument.

source: https://www.tutorialspoint.com/cprogramming/c_functions.htm

PASS BY VALUE

```c
#include <stdio.h>
void swap(int x, int y);
int  main() {
    int a = 100;
    int b = 200;
    printf("Before swap, value of a : %d\n", a);
    printf("Before swap, value of b : %d\n", b);
    swap(a, b);
    printf("After swap, value of a : %d\n", a);
    printf("After swap, value of b : %d\n", b);
    return 0;
}
void swap(int x, int y) {
    int temp;
    temp = x;     /* save the value of x */
    x    = y;     /* put y into x */
    y    = temp;  /* put temp into y */
    return;
}
```

source: https://www.tutorialspoint.com/cprogramming/c_functions.htm

PASS BY REFERENCE

```c
#include <stdio.h>
void swap(int *x, int *y);
int main() {
    int a = 100;
    int b = 200;
    printf("Before swap, value of a : %d\n", a);
    printf("Before swap, value of b : %d\n", b);
    swap(&a, &b);
    printf("After swap, value of a : %d\n", a);
    printf("After swap, value of b : %d\n", b);
    return 0;
}
void swap(int *x, int *y) {
    int temp;
    temp = *x;   /* save the value at address x */
    *x   = *y;   /* put y into x */
    *y   = temp; /* put temp into y */
    return;
}
```

source: https://www.tutorialspoint.com/cprogramming/c_functions.htm

POINTERS

- A pointer is a variable whose value is the address in memory of another variable
- You need them to do dynamic memory allocation in the heap
- Dynamic memory allocation is important when you deal with data of big size that don’t fit on the stack

PASSING POINTERS TO FUNCTION (AS OUTPUT)

```c
#include <stdio.h>
#include <time.h>
void getSeconds(unsigned long *par);
int main() {
    unsigned long sec;
    getSeconds(&sec);
    printf("Number of seconds: %ld\n", sec);
    return 0;
}
void getSeconds(unsigned long *par) {
    *par = time(NULL);
    return;
}
```

PASSING POINTERS TO FUNCTION (AS ARRAY)

```c
#include <stdio.h>
double getAverage(int *arr, int size);
int main() {
    int    balance[5] = {1000, 2, 3, 17, 50};
    double avg;
    avg = getAverage(balance, 5);
    printf("Average value is: %f\n", avg);
    return 0;
}
double getAverage(int *arr, int size) {
    int    i, sum = 0;
    double avg;
    for (i = 0; i < size; ++i) {
        sum += arr[i];
    }
    avg = (double)sum / size;
    return avg;
}
```

PASSING POINTERS TO FUNCTION (AS STRING)

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
void concatOneX(char *output, char *src);
int main(void) {
    char *input = “Something";
    char *output;
    concatOneX(output, input);
    printf("Length: %d\n", strlen(output));
    printf("%s\n", output);
}
```

PASSING POINTERS TO FUNCTION (AS STRING)

```c
void concatOneX(char *output, char *src) {
    int   size    = strlen(src) + 1;
    char *sOutput = malloc(size * sizeof(char));
    if (sOutput == NULL) {
        printf("Something wrong\n”);
        exit(EXIT_FAILURE);
    }
    sOutput = strcpy(sOutput, src);
    if (sOutput == NULL) {
        printf("Something wrong\n”);
        exit(EXIT_FAILURE);
    }
    char *errorCheck = strcat(sOutput, “X");
    if (errorCheck == NULL) {
        printf("Something wrong\n”);
        exit(EXIT_FAILURE);
    }
    strcpy(output, sOutput);
    free(sOutput);
}
```

PASSING POINTERS TO FUNCTION (REFACTORING)

```c
void checkFatal(char *ptr) {
    if (ptr == NULL) {
        printf("Something wrong\n”);
        exit(EXIT_FAILURE);
    }
}
void concatOneX(char *output, char *src) {
    int   size    = strlen(src) + 1;
    char *sOutput = malloc(size * sizeof(char));
    checkFatal(sOutput);
    sOutput = strcpy(sOutput, src);
    checkFatal(sOutput);
    char *errorCheck = strcat(sOutput, “X");
    checkFatal(errorCheck);
    strcpy(output, sOutput);
    free(sOutput);
}
```

PASSING POINTERS TO FUNCTION (ALTERNATIVE)

```
char* concatOneX(char *src) {
    int   size     = strlen(src) + 1;
    char *sOutput = malloc(size * sizeof(char));
    checkFatal(sOutput);
    sOutput = strcpy(sOutput, src);
    checkFatal(sOutput);
    char *errorCheck = strcat(sOutput, “X");
    checkFatal(errorCheck);
    return sOutput;
}
```

PASSING POINTERS TO FUNCTION (ALTERNATIVE)

```c
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
char* concatOneX(char *src);
void checkFatal(char *ptr);
int main(void) {
    char *input = “Something";
    char *output = concatOneX(input);
    printf("Length: %d\n", strlen(output));
    printf("%s\n", output);
    free(output);
}
```

STRUCT

- A `struct` is another user defined data type available in C that allows to combine data items of different kinds.
- They can be use to express more complex data, to model better our reality

STRUCT

- A `struct` is another user defined data type available in C that allows to combine data items of different kinds.
- They can be use to express more complex data, to model better our reality

STRUCT

```
#include <stdio.h>
#include <string.h>
struct Book {
   char  title[50];
   char  author[50];
   char  subject[100];
   int   book_id;
};
int main( ) {
   struct Book Book1;
   struct Book Book2;
   strcpy( Book1.title, "C Programming”);
   strcpy( Book1.author, "Nuha Ali");
   strcpy( Book1.subject, "C Programming Tutorial”);
   Book1.book_id = 6495407;
   strcpy( Book2.title, "Telecom Billing”);
   strcpy( Book2.author, "Zara Ali”);
   strcpy( Book2.subject, "Telecom Billing Tutorial");
   Book2.book_id = 6495700;
   printf( "Book 1 title : %s\n", Book1.title);
   printf( "Book 1 author : %s\n", Book1.author);
   printf( "Book 1 subject : %s\n", Book1.subject);
   printf( "Book 1 book_id : %d\n", Book1.book_id);
   printf( "Book 2 title : %s\n", Book2.title);
   printf( "Book 2 author : %s\n", Book2.author);
   printf( "Book 2 subject : %s\n", Book2.subject);
   printf( "Book 2 book_id : %d\n", Book2.book_id);
   return 0;
}
```

STRUCT

```c
#include <stdio.h>
#include <string.h>
struct Book {
    char title[50];
    char author[50];
    char subject[100];
    int  book_id;
};
void printBook(struct Book book, int n) {
    printf("Book %d title : %s\n", n, book.title);
    printf("Book %d author : %s\n", n, book.author);
    printf("Book %d subject : %s\n", n, book.subject);
    printf("Book %d book_id : %d\n", n, book.book_id);
}
int main() {
    struct Book Book1;
    struct Book Book2;
    strcpy(Book1.title, "C Programming”);
    strcpy(Book1.author, "Nuha Ali”);
    strcpy(Book1.subject, "C Programming Tutorial”);
    Book1.book_id = 6495407;
    strcpy(Book2.title, "Telecom Billing”);
    strcpy(Book2.author, "Zara Ali”);
    strcpy(Book2.subject, "Telecom Billing Tutorial");
    Book2.book_id = 6495700;
    printBook(Book1, 1);
    printBook(Book2, 2);
    return 0;
}
```

STRUCT

```c
#include <stdio.h>
#include <string.h>
struct Book {
    char title[50];
    char author[50];
    char subject[100];
    int  book_id;
};
void printBook(struct Book book, int n) {
    printf("Book %d title : %s\n", n, book.title);
    printf("Book %d author : %s\n", n, book.author);
    printf("Book %d subject : %s\n", n, book.subject);
    printf("Book %d book_id : %d\n", n, book.book_id);
}
struct Book buildBook(char *title, char *author, char *subject, int book_id) {
    struct Book book;
    strcpy(book.title, "C Programming");
    strcpy(book.author, "Nuha Ali");
    strcpy(book.subject, "C Programming Tutorial");
    book.book_id = 6495407;
    return book;
}
int main() {
    struct Book Book1 = buildBook(
      "C Programming",
      "Nuha Ali",
      "C Programming Tutorial",
      6495407
    );
    struct Book Book2 = buildBook(
      "Telecom Billing",
      "Zara Ali",
      "Telecom Billing Tutorial",
      6495700
    );
    printBook(Book1, 1);
    printBook(Book2, 2);
    return 0;
}
```

STRUCT

```c
#include <stdio.h>
#include <string.h>
struct Book {
    char title[50];
    char author[50];
    char subject[100];
    int  book_id;
};
void printBook(struct Book book, int n) {
    printf("Book %d title : %s\n", n, book.title);
    printf("Book %d author : %s\n", n, book.author);
    printf("Book %d subject : %s\n", n, book.subject);
    printf("Book %d book_id : %d\n", n, book.book_id);
}
void buildBook(struct Book* book, char *title, char *author, char *subject, int book_id) {
    strcpy(book->title, "C Programming");
    strcpy(book->author, "Nuha Ali");
    strcpy(book->subject, "C Programming Tutorial");
    book->book_id = 6495407;
}
int main() {
    struct Book* Book1 = malloc(sizeof(struct Book));
    buildBook( Book1,
      "C Programming",
      "Nuha Ali",
      "C Programming Tutorial",
      6495407
    );
    struct Book* Book2 = malloc(sizeof(struct Book));
    buildBook( Book2,
      "Telecom Billing",
      "Zara Ali",
      "Telecom Billing Tutorial",
      6495700
    );
    printBook(*Book1, 1);
    printBook(*Book2, 2);
    free(Book1);
    free(Book2);
    return 0;
}
```

SOME COVID BEST PRACTICES BEFORE WE LEAVE

- Disinfect table and chair
- Maintain your distance to others
- Wash or sanitise your hands
- Respect guidelines and restrictions outside