Opgave 1

Write four different C statements that each add 1 to integer variable z.

```
#include <stdio.h>
int main(){
    int z = 0;
    ++z;
    z++;
    z+=1;
    z = z+1;
}
```

Opgave 2

Identify and correct the errors in each of the following. [Note:There may be more than one error in each piece of code.]

Assuming both pieces of code are placed in a correct main function:

```
    int x = 1, product = 0;
        while ( x <= 10)
    {
        product *= x;
        ++x;
    }
```

```

    while( x <= 100)
        total =+ x;
    ++x;
```

Errors in a:
While loop does nothing, except make x = 11.
Multiplying with 0 10 times is still 0.

Errors in b:
x and total is undeclared.
No {} for the while, meaning we get stuck in loop if x is under 100.
=+ is equal to = when x is positive.

Opgave 3

What does the following program print:

```
#include<stdio.h>

 intmain(void)
 {
    int y;
    int x = 1;
    int total = 0;

  while( x <= 5) {
        y = x * x * x;
        printf( "%d\n", y );
       total += y;
        ++x;
  } // end while

 printf( "The total is %d\n", total );
 } // end main
```

Technical we would get an error since intmain is wrong, however if that is fixed:

```
1
8
27
64
125
The total is 225
```

Opgave 4

Write a single pseudocode statement (or C statement) that indicates each of the following:

a) Display the message ”Enter two numbers”.

b) Assign the sum of variables x, y, and z to variable p.

c) The following condition is to be tested in an if...elseselection statement: The current value of variable mis greater than twice the current value of variable v.

d) Obtain values for variables s, r, and t from the keyboard.

```c
#include<stdio.h>
int x,y,z,p,s,r,t, mis = 0, v = 0;

int main(){
    //a
    printf("Enter two numbers");

   p = x + y + z;

   if(mis > v * 2){
       printf("mis is greater than twice the current value of variable v")
   }
   else{
       printf("mis isn't greater than twice the current value of variable v")
   }

   scanf("%d%d%d", &s, &r, &t);
}
```

Opgave 5

What does the following program print:

```c
#include <stdio.h>

int main( void)
{
    int x = 1,  total  = 0,  y;

    while( x  <= 10) {
        y  =  x  *  x;
        printf (  "%d\n", y  ) ;
        total  += y;
        ++x;
    }

    printf ("Total  is  %d\n",  total ) ;
    return 0;
}
```

```
1
4
9
16
25
36
49
64
81
100
Total is 385
```

Opgave 6

What does the following program print:

```c
#include <stdio.h>

int main( void)
{
    int outer_count = 1;

    while ( outer_count <= 10 ) {
        int inner_count = 1;
        while (inner_count <= outer_count) {
            printf( "*" );
            inner_count++;
        } // end inner while

        printf( "\n" );
        outer_count++;

    } // end outer while
}// end main
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

Opgave 7

What does the following program print:

```c
#include <stdio.h>

int main(void)
{
    int i;
    int j;
    for(i=20, j=300; i <=j; i+=2, j -= 2){
        printf("%d\n",i+j);
    }
    return 0;
}
```

i+j will always be 320, so the program will always print 320.
Where will be i = 162 and j = 158 in the end, because when i and j = 160, we run again for the last time and add 2 to i and subtract 2 from j.
160 160 = true
162 158 = false

Opgave 8

Answer following questions:
How many times is a do while loop guaranteed to loop?
It will always run once since you do something and then check, if we should run again.

In switch statement what keyword covers unhandled possibilities?

```c
switch(char/int/enum/short){ //we use
case value: c code
case value: c code
default: c code // the answer
}
```

What separates particular elements at function parameter list?
function(int a, int b) = parameters
function(1, 2) = arguments
Answer is comma

Opgave 9

What does the following program print:

```c
#include <stdio.h>
int main()
{
    int
    t[4]=  {10,20,30,40};
    int * ad [4];
    int i;
    for (i=0;i<4;i++)
        ad[i] = t+i;  //Set all pointer in ad to point to t.
    for (i=0;i<4;i++)
         printf("%d ", *ad[i]);
     printf("\n");
     printf("%d %d \n", * (ad[1]+1), * ad[1]+1);
     //First print is equal to *ad[1+1]
     //Second print is equal to take what is at ad[1] and add 1 to it
    return 0;
}
```

```
10 20 30 40
30 21
```

Note to self:
* means get the thing at the place the pointer is pointing to.
Without *: change the value/address of the pointer.

Opgave 10

What does the following program print:

```
1  #include <stdio.h>
2  int main()
3  {
4      int a[2][3] = {1, 2, 3, 4, 5}; // All undefined values is changed to 0.
5      int i = 0, j = 0;
6      for (i = 0; i < 2; i++) {
7          for (j = 0; j < 3; j++) {
8              printf("%d", a[i][j]);
9              return 0;
10         }
11     }
12 }
```

Line 4 is badly written:

```
1  int a[2][3] = {{1,2,3},{4,5,0}};
```

We print all values out in order:
[0][0],[0][1],[0][2],[1][0],[1][1],[1][2]
123450

Opgave 11

What does the following program print:

```
 1  #include <stdio.h>
 2
 3  void mystery(int *ptra, int *ptrb)
 4  {
 5      int *temp;
 6      temp = ptrb;
 7      ptrb = ptra;
 8      ptra = temp;
 9  }
10
11  int main()
12  {
13      int a=2018, b=0, c=4, d=42;
14      mystery(&a, &b);
15      if (a < c)
16          mystery(&c, &a);
17      mystery(&a, &d);
18      printf(”%d\n”, a);
19      return 0;
20  }
```

Mystery does nothing, so the result is 2018
Short answer: C does not allow a user to mess with memory in the sence of swapping memory directly, since 0000 need to come before 0004.
Therefore one should swap the values that are placed on the addresses instead, which can be done by using pointers. Imagine you have 4 houses on a row. number 1, 3, 5 and 7, which is qual to addresses. Smith (3) and Johnson (5) wants to swap houses. Smith and Johnson is equal to variables.
Instead of swapping the position of the houses, so we have 1, 5, 3 and 7, which is going to fuck up PostNord. Smith and Johnson swap themselves, so we have 1, Johnson, Smith and 7.
Correct way of swapping:

```
1  void mystery(int *ptra, int *ptrb)
2  {
3      int temp = *ptra;
4      *ptra = *ptrb;
5      *ptrb = temp;
6  }
```

This will make swapping possible, so we get 42. because a and d have swapped in line 17.

Opgave 12

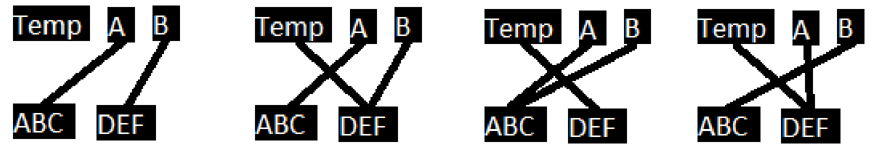
If a variable has to store the address of the address of a character, what must be the type of its definition?
Short answer char**.
Long answer: If we have the code from question 11, and modify it to work with chars instead, so we can swap pointers.

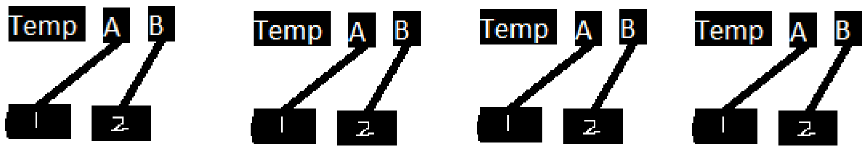
```c
#include <stdio.h>

void mystery(char **ptra, char **ptrb)
{
    char *temp = *ptrb;
    *ptrb = *ptra;
    *ptra = temp;
}

int main()
{
    char *a=”abc”, *b=”def”;
    mystery(&a, &b);
    printf(”%s\n”, a);
    return 0;
}
```

If we compare this to question 11, A and B are now pointer to places in memory instead of memory itself. In the top one, we change the pointer values, and in the bottom we change the values.





Opgave 13

What does the following program print:

```
#include <stdio.h>
union myUnion {
    int x, y;
};

int main(){
    union myUnion u;
    u.x = 2;
    u.y = 10;
    printf("X: %d, Y: %d\n", u.x, u.y);
    return 0;
}
```

X: 10, Y: 10
This is because a union stores everything on the same memory space.
For this to work use a struct. However unions are smart as long as we are only using 1 thing at a time.
Also use:

```
typedef union {
    int x, y;
} myUnion;
myUnion u;
```

Opgave 14

An array with four 4-byte integers is defined as int array[4] and is stored starting at position 100 in memory. The following code is executed:

```
array[0] = 20;
array[1] = 30;
array[2] = 10;
array[3] = 0;
```

In which memory address is stored the first element of the array?
The different elements are saved at position 100-103, 104-107, 108-111 and 112-115.

Opgave 15

What does the following program print:

```
1  struct Course {
2      int id;
3      char *name;
4  };
5  struct Student {
6      char *name;
7      struct Course courses[3];
8  };
9
10 int main(){
11     struct Student student1 = {”Alice”},
12             student2 = {”Bob”};
13     struct Course course1 = {62409, ”C Programming”},
14             course2 = {62410, ”C++ Programming”},
15             course3 = {62411, ”Java”};
16     student1.courses[0] = course1;
17     student1.courses[1] = course2;
18     student1.courses[2] = course3;
19     student2.courses[0] = course1;
20     student2.courses[2] = student1.courses[2];
21     printf(”%s”,student2.courses[2].name);
22     return 0;
23 }
```

Not really much since we hve no #include <stdio.h>, but if we had:
Everything we need is on line 11, 12, 15, 18, 20 and 21. Everything else is filler.
Line 11 and 12: We create Alice and Bob.
Line 15: course3 is Java.
Line 18: Alices course in slot 2 is Java. Line 20: Bobs course in slot 2 is the same as Alices slot 2 (Java).
Line 21: We print Bobs course in slot 2 (Java).

Opgave 16

What does the following program print:

```c
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *next;
};
void printList (struct Node *n) {
    while (n != NULL) {
        printf(" %d ", n->data);
        n = n->next;
    }
}
int main(int argc, char** argv) {
    struct Node* item1 = NULL;
    struct Node* item2 = NULL;
    struct Node* item3 = NULL; // allocate 3 nodes on the heap
    item1 = (struct Node*) malloc(sizeof (struct Node));
    item2 = (struct Node*) malloc(sizeof (struct Node));
    item3 = (struct Node*) malloc(sizeof (struct Node));
    item1->data = 3;
    item1->next = item2;
    item2->data = 2; //assign data to second node
    item2->next = item3;
    item3->data = 1; //assign data to third node
    item3->next = NULL;
    printList (item1);
    return (EXIT_SUCCESS);
}
}
```

Will print " %d %d %d " or " 3 2 1 " Good time to talk about calloc, malloc, realloc and free.
Malloc: Allocate memory block or bunch of memory. ptr = (int*) malloc(100 * sizeof(int));
Calloc: This is used to allocate cells or partitioned memory block. Sets everything to 0. ptr = (int*) calloc(100, sizeof(int));
Realloc: Allocate memory if we run out. ptr = realloc(ptr, 150);
Free: frees the memory so we can use it for something else. free(ptr);

Opgave 17

Answer each of the following questions:
a) What is the difference between passing arguments by value and passing arguments by reference?
If you pass an argument by value, then that value will cease to exist when you exit from the fuction and all changes will be discarded.
If you pass an argument by reference (pointer/variable), you can change things in the function, and changes will not be discarded.

b) What values does the rand function generate?
rand(void) is a function that return a pseudo-random number from 0 to at least 32767, (which is 2 to the power of 15 or a 2 byte value.) based on a seed, like in Slay the Spire.

c) How do you randomize a program? How do you scale or shift the values produced by the rand function?
We can use srand(time(NULL)) or srand(unsigned int seed) to give a seed based on the current time. However we can't change RAND_MAX, unless we want to break the compiler.
To change the numbers to a certain threeshold, we can use modulus. E.G: rand() % 9 + 10 gives a number from 10 to 19.

d) What is a recursive function?
A function that calls itself. This can go wrong very fast, if you forget to program an exit statement.
E.G all your local variables are located on the stack, while all global/static variables and allocated memory is on the heap.
All we create more variables on the stack, we are going to smash into the heap at some point. And then... well buckle up, because then it is going to be just like Titanic.