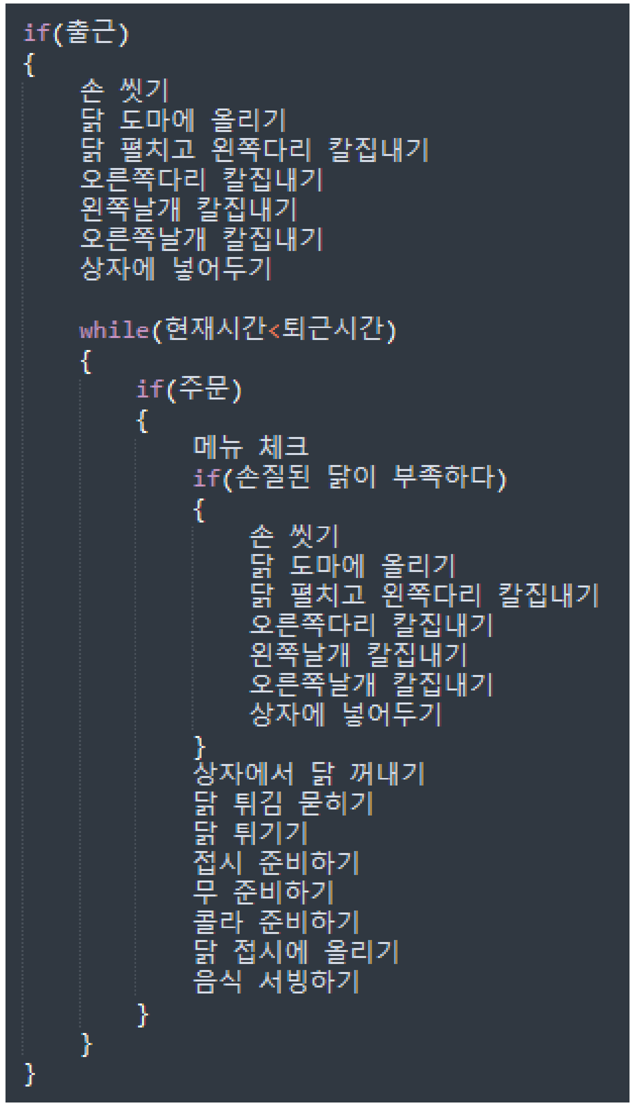
Function

DIGIPEN DRAGONS

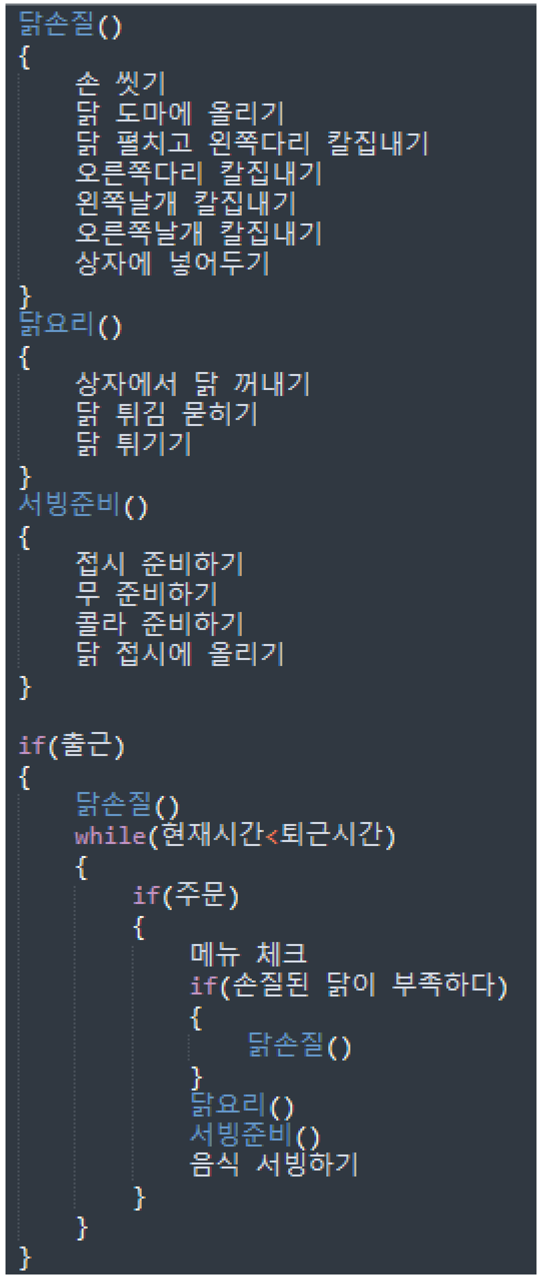
Jinwoo Choi

Function(함수)

Function(함수)



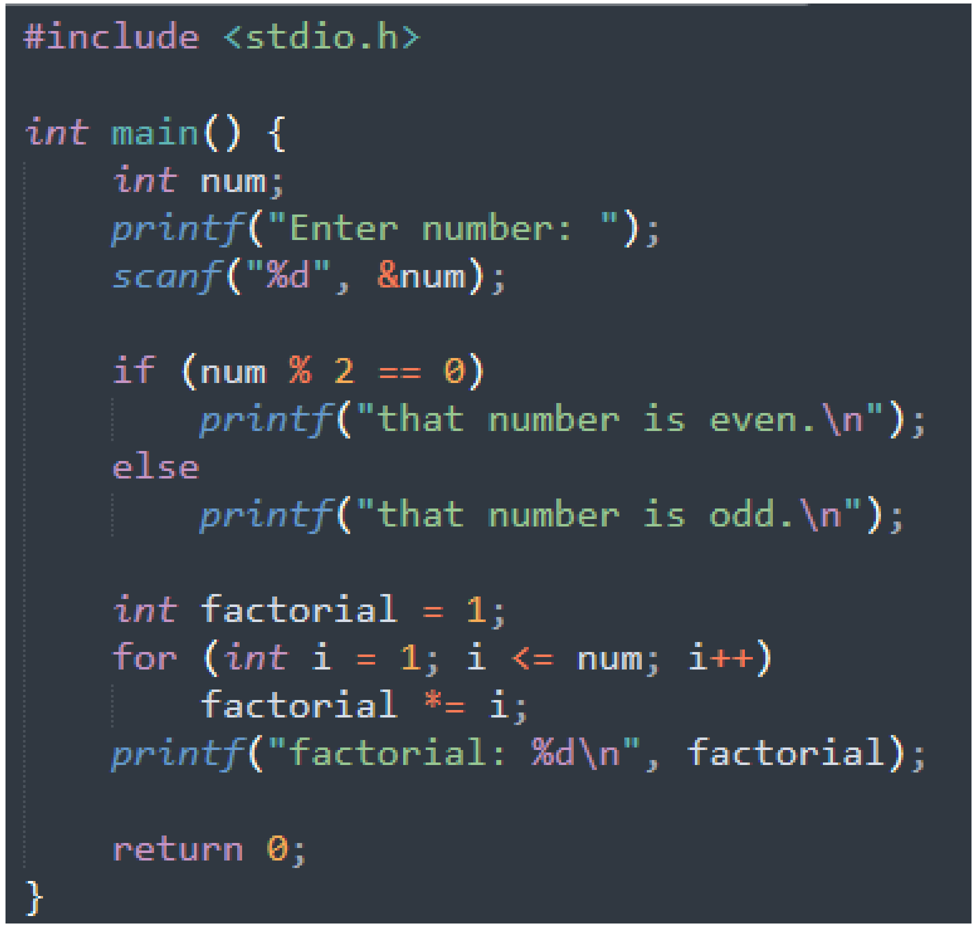
```
if(출근)
{
    손 씻기
    닭 도마에 올리기
    닭 펼치고 왼쪽다리 칼집내기
    오른쪽다리 칼집내기
    왼쪽날개 칼집내기
    오른쪽날개 칼집내기
    상자에 넣어두기

    while(현재시간<퇴근시간)
    {
        if(주문)
        {
            메뉴 체크
            if(손질된 닭이 부족하다)
            {
                손 씻기
                닭 도마에 올리기
                닭 펼치고 왼쪽다리 칼집내기
                오른쪽다리 칼집내기
                왼쪽날개 칼집내기
                오른쪽날개 칼집내기
                상자에 넣어두기
            }
            상자에서 닭 꺼내기
            닭 튀김 묻히기
            닭 튀기기
            접시 준비하기
            무 준비하기
            콜라 준비하기
            닭 접시에 올리기
            음식 서빙하기
        }
    }
}
```



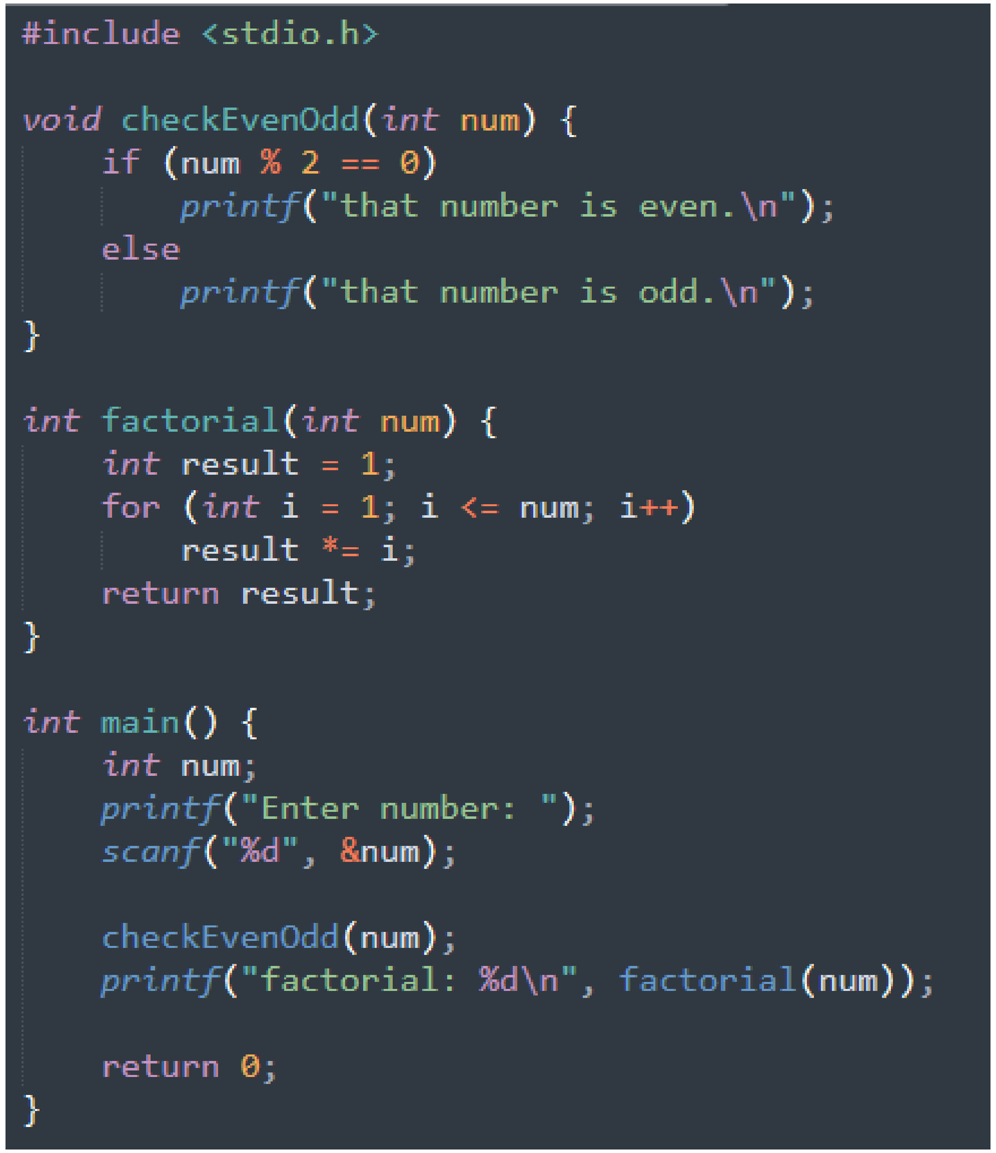
```
닭손질()
{
    손 씻기
    닭 도마에 올리기
    닭 펼치고 왼쪽다리 칼집내기
    오른쪽다리 칼집내기
    왼쪽날개 칼집내기
    오른쪽날개 칼집내기
    상자에 넣어두기
}
닭요리()
{
    상자에서 닭 꺼내기
    닭 튀김 묻히기
    닭 튀기기
}
서빙준비()
{
    접시 준비하기
    무 준비하기
    콜라 준비하기
    닭 접시에 올리기
}

if(출근)
{
    닭손질()
    while(현재시간<퇴근시간)
    {
        if(주문)
        {
            메뉴 체크
            if(손질된 닭이 부족하다)
            {
                닭손질()
            }
            닭요리()
            서빙준비()
            음식 서빙하기
        }
    }
}
```

- Readablility
- Maintainability
- Reusability
- Scalability

Function



```c
#include <stdio.h>

int main() {
    int num;
    printf("Enter number: ");
    scanf("%d", &num);

    if (num % 2 == 0)
        printf("that number is even.\n");
    else
        printf("that number is odd.\n");

    int factorial = 1;
    for (int i = 1; i <= num; i++)
        factorial *= i;
    printf("factorial: %d\n", factorial);

    return 0;
}
```



```c
#include <stdio.h>

void checkEvenOdd(int num) {
    if (num % 2 == 0)
        printf("that number is even.\n");
    else
        printf("that number is odd.\n");
}

int factorial(int num) {
    int result = 1;
    for (int i = 1; i <= num; i++)
        result *= i;
    return result;
}

int main() {
    int num;
    printf("Enter number: ");
    scanf("%d", &num);

    checkEvenOdd(num);
    printf("factorial: %d\n", factorial(num));

    return 0;
}
```

Function Definition

```
returnType FunctionName(parameters)
{
    // Function body
    return returnValue;
}
```

```
int MultiplyXY(int X, int Y)
{
    int result = X * Y;
    return result;
}
```

- returnType: Specifies the data type the function returns. (= returnValue's type)
  - int, void, double … etc
- FunctionName: Unique name of the function
  - sum, multiply, displayText, doSomething … etc
- parameters: Input values the function accepts
  - (int x, int y), (float radius), (), (char c) … etc
- Function Body: Code block that executes the function logic

Functions

```
void PrintHello(void)
{
    printf("Hello World!\n");
    return;
}

int main(void)
{
    PrintHello();
    return 0;
}
```

```
void PrintMultipleHello(int count)
{
    for(int i=0; i<count; i++)
    {
        PrintHello();
    }
    return;
}
int main(void)
{
    int cnt = 5;
    PrintMultipleHello(cnt);
    return 0;
}
```

Scope and Lifetime

- Local Variable(지역변수)
- Global Variable(전역변수)

- Keywords
  - Static
  - Extern

Local Variable

Only used inside the function

```
int MultiplyXY(int X, int Y)
{
    int result_func = X * Y;
    return result_func;
}


int main(void)
{
    int a = 9, b = 6;
    int result_main = MultiplyXY(a, b);
    //print
    return 0;
}
```

<Accessible>

```
printf("%d\n", a)
printf("%d\n", b)
printf("%d\n", result_main)
```

<Inaccessible>

```
printf("%d\n", X)
printf("%d\n", Y)
printf("%d\n", result_func)
```

int a, b, X, Y, result_func, result_main = Local Variable

Global Variable

Available throughout the file where it is defined
To access it in other files, the [extern] keyword must be used

```
int X, Y, result_func;
int MultiplyXY()
{
    result_func = X * Y;
    return result_func;
}
int main(void)
{
    X = 9;
    Y = 6;
    int result_main = MultiplyXY();
    //print
    return 0;
}
```

```
<Accessible>
printf("%d\n", X)
printf("%d\n", Y)
printf("%d\n", result_main)
printf("%d\n", result_func)
<Inaccessible>
```

int X, Y, result_func = Global Variable
int result_main = Local Variable

Extern

Refers to an externally declared global variable

testExtern.h

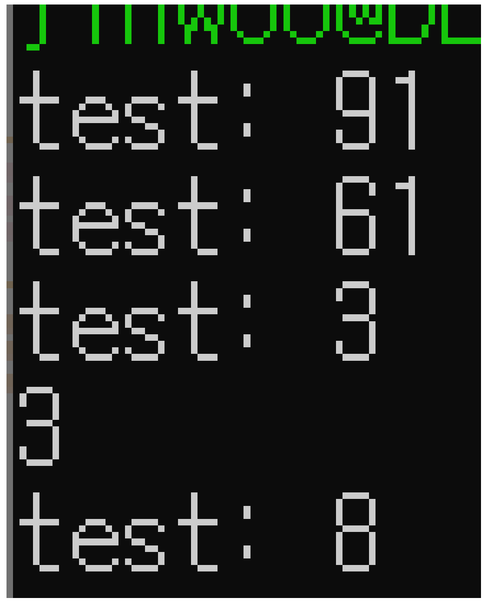
```
void testExtern(void);
```

testExtern.c

```
extern int Z;
void testExtern(void)
{
    ~~printf(“%d\n”, X);~~  //X
    ~~printf(“%d\n”, Y);~~  //X
    printf(“%d\n”, Z);  //O
    Z+=5;
}
```

test.c

```
int X = 91;
int Y = 61;
int Z = 3;
int main(void)
{
    printf("test: %d\n", X);
    printf("test: %d\n", Y);
    printf("test: %d\n", Z);
    testExtern();
    printf("test: %d\n", Z);
}
```



```
test: 91
test: 61
test: 3
3
test: 8
```

gcc -std=c11 -pedantic-errors -Wstrict-prototypes -Wall -Wextra -Werror test.c testExtern.c

Static Global Variable

Available only within the same file

```
static int X, Y, result_func;
int MultiplyXY()
{
    result_func = X * Y;
    return result_func;
}
int main(void)
{
    X = 9;
    Y = 6;
    int result_main = MultiplyXY();
    //print
    return 0;
}
```

<Accessible>

```
printf("%d\n", X)
printf("%d\n", Y)
printf("%d\n", result_main)
printf("%d\n", result_func)
```

<Inaccessible>

inaccessible from other files

ex) if static global variable is defined in test.c
    in other.c can't use that static global variable

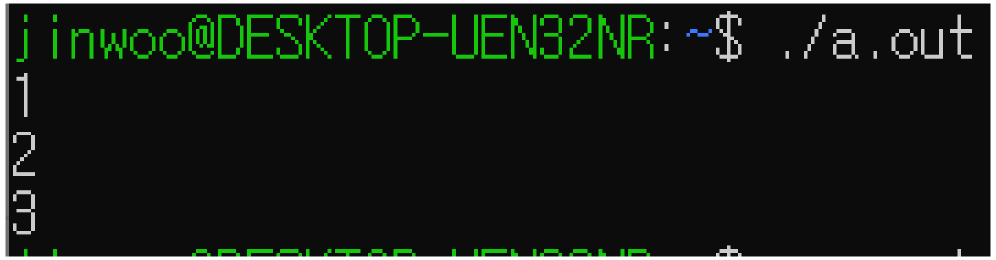
static int X, Y, result_func = Static Global Variable
int result_main = Local Variable

Static Local Variable

Retains value within the function

```
int Counter()
{
    static int count = 0; // initialized only once
    count++;
    return count;
}

int main(void)
{
    printf("%d\n", Counter());
    printf("%d\n", Counter());
    printf("%d\n", Counter());
    return 0;
}
```

Output:



```
jinwoo@DESKTOP-UEN32NR:~$ ./a.out
1
2
3
```

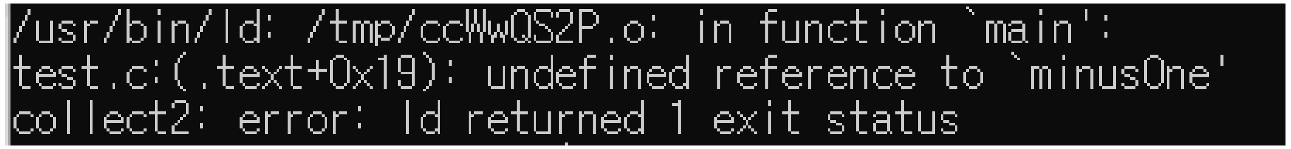
static int count= Static Local Variable

inline

Reduces function call overhead

```
inline int MinusOne(int num)
{
    return num - 1;
}

int main(void)
{
    int number = 113;
    number = MinusOne(number);
    return 0;
}
```

```
int main(void)
{
    int number = 113;
    number = number - 1;
    return 0;
}
```



```
/usr/bin/ld: /tmp/ccWwQS2P.o: in function `main':
test.c:(.text+0x19): undefined reference to `minusOne'
collect2: error: ld returned 1 exit status
```

```
gcc -std=c11 -pedantic-errors -Wall -Wextra -Werror test.c -O2
```

-O2 : More Optimizing
-O3 : More and more Optmizing

inline

```
inline void Print100(void)
{
    for(int i=0; i<100; i++)
    {
        printf("%d\n",i);
    }
}
int main(void)
{
    Print100();
    return 0;
}
```

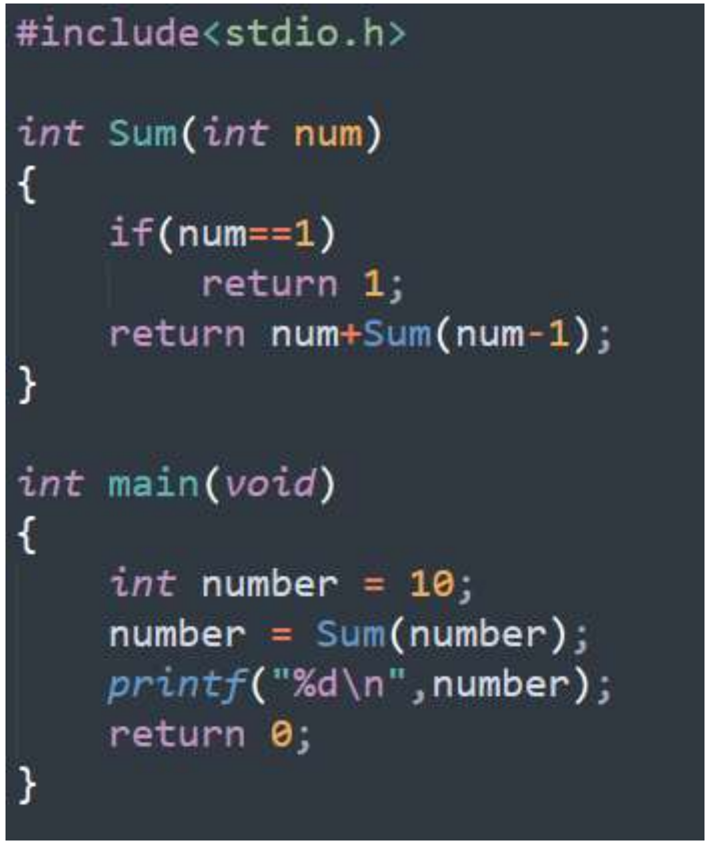
```
inline int MinusOne(int num)
{
    return num - 1;
}

int main(void)
{
    int number = 113;
    number = MinusOne(number);
    number = MinusOne(number);
    number = MinusOne(number);
    return 0;
}
```

Using inline when,
function itself is too big
or
it repeatedly called

increases the size of the executable.

-> doesn't matter for small program
-> Hard to load function into CPU cache
Might cause **CPU cache miss** in big program

Recursive

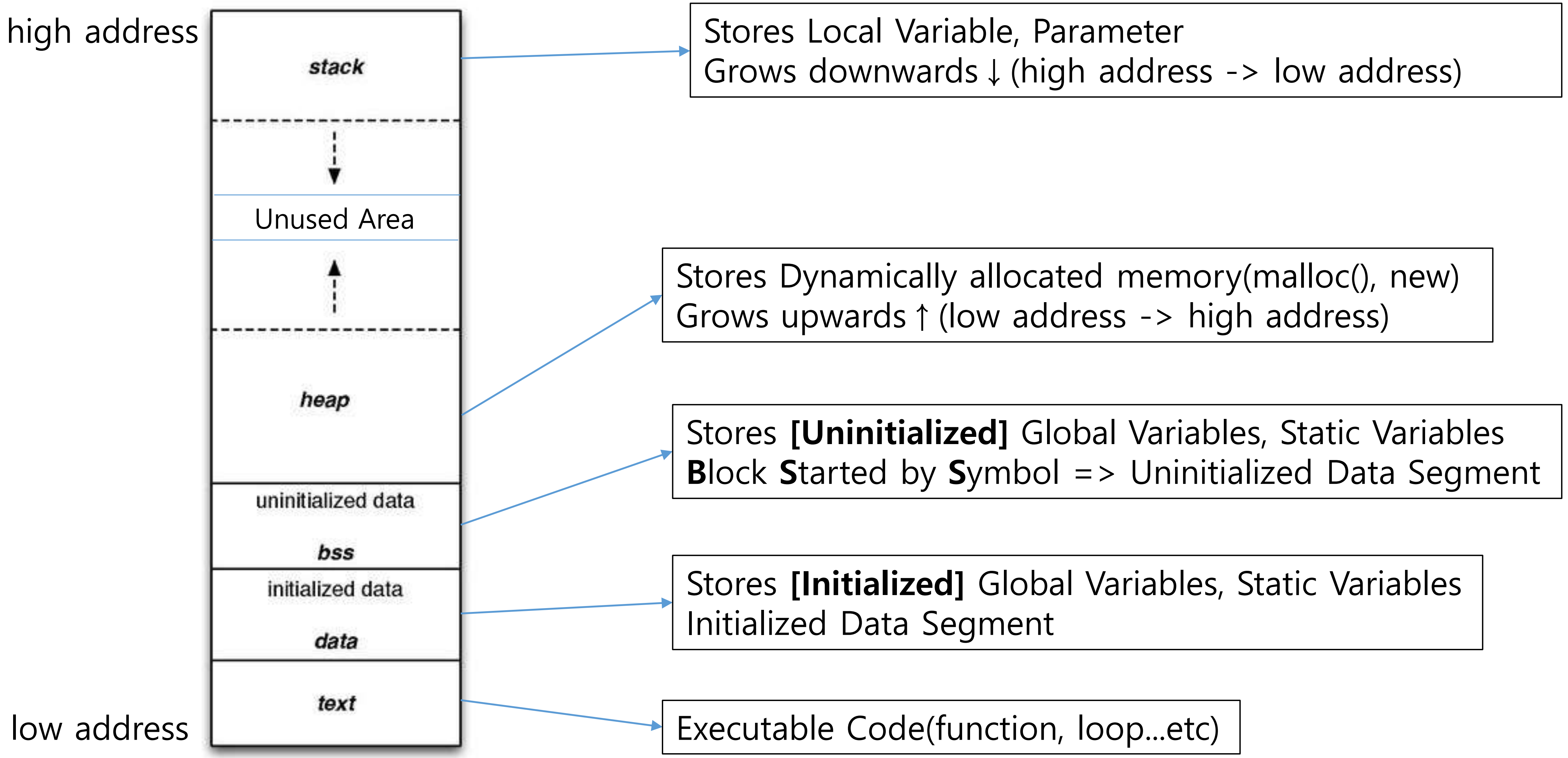


```c
#include<stdio.h>

int Sum(int num)
{
    if(num==1)
        return 1;
    return num+Sum(num-1);
}

int main(void)
{
    int number = 10;
    number = Sum(number);
    printf("%d\n",number);
    return 0;
}
```

(num==1) : Base Case

```
• number=1 :
    if(num==1)
         return 1;

• number=2 :
    return 2+Sum(2-1);
    -> if(num==1)
           return 1;
    return 2+1;

• number=3 :
    return 3+Sum(3-1);
    -> return 2+Sum(2-1);
         -> if(num==1)
            return 1;
         return 2+1;
    return 3+3;
```

Memory



Stores Local Variable, Parameter
Grows downwards↓(high address -> low address)

Stores Dynamically allocated memory(malloc(), new)
Grows upwards↑(low address -> high address)

Stores **[Uninitialized]** Global Variables, Static Variables
Block **S**tarted by **S**ymbol => Uninitialized Data Segment

Stores **[Initialized]** Global Variables, Static Variables
Initialized Data Segment

Executable Code(function, loop...etc)

LAB – Exponentiation

- Create a file named 'Exponentiation_YourName.c'.
- Your program should prompt the user to enter a base(x) and an exponent(n).
- Implement a function to calculate x^n using:
  - Recursion
  - Iteration(loop-based)
  - ex) double PowerRecursive(double x, int n)

    double PowerIterative(double x, int n)
- Display the result of exponentiation for both methods.