CyberSecurity: Principle and Practice

BSc Degree in Computer Science
2025-2026
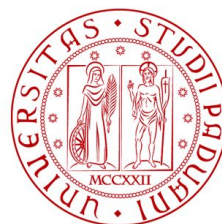
Bonus Lesson: C Recap

Prof. Mauro Conti
Department of Mathematics
University of Padua
mauro.conti@unipd.it
http://www.math.unipd.it/~conti/

Teaching Assistants
Giulio Umbrella
giulio.umbrella@phd.unipd.it
Francesco De Giudici
francesco.degiudici@studenti.unipd.it

UNIVERSITÀ DEGLI STUDI DI PADOVA
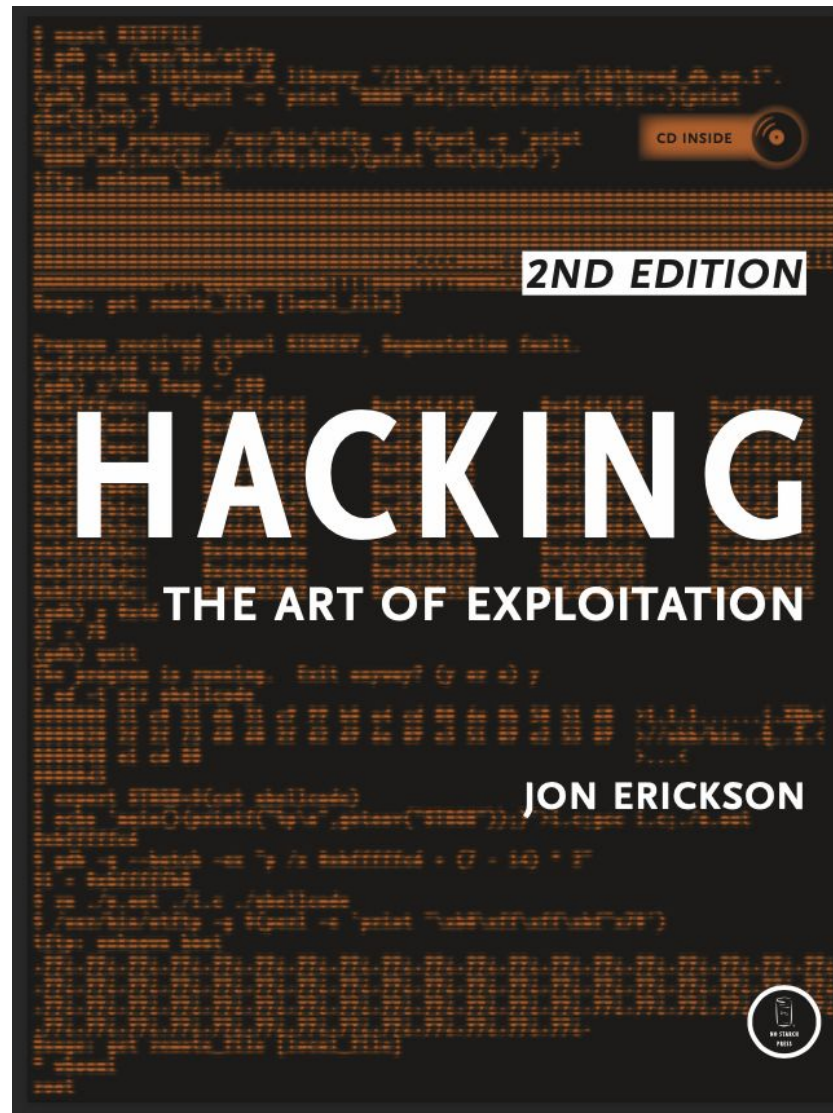
SPRITZ SECURITY & PRIVACY RESEARCH GROUP

DIPARTIMENTO MATEMATICA

Overview of C

- C is developed by Dennis Ritchie 1972

- C is a structured programming language

- C supports functions that enables easy maintainability of code, by breaking large file into smaller modules

- Comments in C provides easy readability

- C is a powerful language

Program structure

A sample C Program

```c
#include<stdio.h>

int main()
{
    --other statements
    // Comments after double slash
}
```

Header files

- The files that are specified in the include section are called header files
- These are precompiled files that have some functions defined in them
- We can call those functions in our program by supplying parameters
- Header file is given an extension .h
- C Source file is given an extension .c

Main function

- This is the entry point of a program
- When a file is executed, the start point is the main function
- From main function the flow goes as per the programmers choice.
- There may or may not be other functions written by user in a program
- Main function is compulsory for any c program

Writing the first program

```c
#include<stdio.h>
int main()
{
    printf("Hello");
    return 0;
}
```

This program prints Hello on the screen when we execute it

Running a C Program

- Type a program
- Save it
- Compile the program: gcc main.c -o output
    This will generate an executable file
- Run the program (Actually the exe created out of compilation will run and not the .c file)
- In different compiler we have different option for compiling and running. We give only the concepts.

- # Single line comment
  - – // (double slash)
  - – Termination of comment is by pressing enter key
- # Multi line comment

  /*….

  …….*/

  This can span over to multiple lines

Data types in C

- Primitive data types
  - int, float, double, char
- Aggregate data types
  - Arrays come under this category
  - Arrays can contain collection of int or float or char or double data
- User defined data types
  - Structures and enum fall under this category.

Variables

- Variables are data that will keep on changing
- Declaration

  <<Data type>> <<variable name>>;

  int a;

- Definition

  <<varname>>=<<value>>;

  a=10;

- Usage

  <<varname>>

  a=a+1;   //increments the value of a by 1

Variable names - Rules

- Should not be a reserved word like int etc..

- Should start with a letter or an underscore(_)

- Can contain letters, numbers or underscore.

- No other special characters are allowed including space

- Variable names are case sensitive
  A and a are different.

- Input
  - scanf("%d",&a);
  - Gets an integer value from the user and stores it under the name "a". Stops at whitespace
  - gets(char *str) reads a line from stdin and stores it into the string pointed to by str

- Output
  - printf("%d",a);
  - Prints the value present in variable a on the screen
  - puts(const char *str) writes a string to stdout but not including the null character.

%f,%g: placeholders for a float or double value
 %d: placeholder for a decimal value (for printing char, short, int values)
 %u: placeholder for an unsigned decimal
 %c: placeholder for a single character
 %s: placeholder for a string value
 %p: placeholder to print an address value

 To print out long type values need to use I prefix:
   %lu: print an unsigned long value
   %lld: print a long long value

Placeholders for specifying the numeric representation
 %x: print value in hexidecimal (base 16)

```
if (condition)
{
    stmt 1;    //Executes if Condition is true
}
else
{
    stmt 2;   //Executes if condition is false
}
```

Conditional statement

```
switch(var)
{
case 1:     //if var=1 this case executes
            stmt;
            break;
case 2:     //if var=2 this case executes
            stmt;
            break;
default:    //if var is something else this will execute
            stmt;
}
```

The syntax of for loop is

```
for(initialisation;condition checking;increment)
{
    set of statements
}

Eg: Program to print Hello 10 times
for(I=0;I<10;I++)
{
    printf("Hello");
}
```

The syntax for while loop

```
while(condition)

{

    statements;

}
```

Eg:

```
a=10;
while(a != 0)
{
    printf("%d",a);
    a--;
}
```

Output: 10987654321

Do While loop

The syntax of do while loop

```
do
 {
    set of statements
 } while(condn);
```

Eg:
```
    i=10;
    do
    {
        printf("%d",i);  i--;
    }  while(i!=0)
```

Output: 10987654321

Operators

- Arithmetic (+,-,*,/,%)
- Relational (<,>,<=,>=,==,!=)
- Logical (&&,||,!)
- Bitwise (&,|)
- Assignment (=)
- Compound assignment(+=,*=,-=,/=, %=,&=,|=)
- Shift (right shift >>, left shift <<)

- Procedure is a function whose return type is void

- Functions will have return types int, char, double, float or even structs and arrays

- Return type is the data type of the value that is returned to the calling point after the called function execution completes

Declaration section

<<Returntype>> funname(parameter list);

Definition section

<<Returntype>> funname(parameter list)
{
    body of the function
}

Function Call

Funname(parameter);

Example function

```c
#include<stdio.h>

void fun(int a);              //declaration
int main()
{
    fun(10);                  //Call
}
void fun(int x)               //definition
{
    printf("%d",x);
}
```

- Actual parameters are those that are used during a function call (value is known)

- Formal parameters are those that are used in function definition and function declaration (value is unknown)

Arrays

- Arrays fall under aggregate data type
- Aggregate – More than 1
- Arrays are collection of data that belong to same data type
- Arrays are collection of homogeneous data
- Array elements can be accessed by its position in the array called as index

- Array index starts with zero
- The last index in an array is num – 1 where num is the no of elements in a array
- int a[5] is an array that stores 5 integers
  - a[0] is the first element where as a[4] is the fifth element
- We can also have arrays with more than one dimension
- float a[5][5] is a two dimensional array. It can store 5x5 = 25 floating point numbers
  - The bounds are a[0][0] to a[4][4]

Strings are arrays of chars -- char c[] = "c string";

- strlen(str) – To find length of string str

- strrev(str) – Reverses the string str as rts

- strcat(str1,str2) – Appends str2 to str1 and returns str1

- strcpy(st1,st2) – copies the content of st2 to st1

- strcmp(s1,s2) – Compares the two string s1 and s2

- strcmpi(s1,s2) – Case insensitive comparison of strings

- Structures are user defined data types
- It is a collection of heterogeneous data
- It can have integer, float, double or character data in it
- We can also have array of structures

```
struct <<structname>>
{
   members;
} element;
```

We can access element.members;

```
struct Person
{
    int id;
    char name[5];

} P1;


P1.id = 1;
P1.name = "vasu";
```

- Pointer is a special variable that stores address of another variable

- Addresses are integers. Hence pointer stores integer data

- Size of pointer = size of int

- Pointer that stores address of integer variable is called as integer pointer and is declared as int *ip;

- Pointers that store address of a double, char and float are called as double pointer, character pointer and float pointer respectively.
- char *cp
- float *fp
- double *dp;
- Assigning value to a pointer
   int *ip = &a; //a is an int already declared

Examples

int a;

a=10;   //a stores 10

int *ip;

ip = &a;   //ip stores address of a (say 1000)

ip :   fetches 1000

*ip   :   fetches 10

* Is called as dereferencing operator

Calling a function with parameters passed   as values

int a=10;
fun(a);

```
void fun(int a)
{
        defn;
}
```

Here fun(a) is a call by value.

Any modification done with in the function is local to it and will not be effected outside the function

Example program – Call by value

```
#include<stdio.h>
void main()

{
    int a=10;
    printf("%d",a);        a=10
    fun(a);
    printf("%d",a);        a=10

}
void fun(int x)

{
    printf("%d",x)         x=10
    x++;
    printf("%d",x);        x=11

}
```
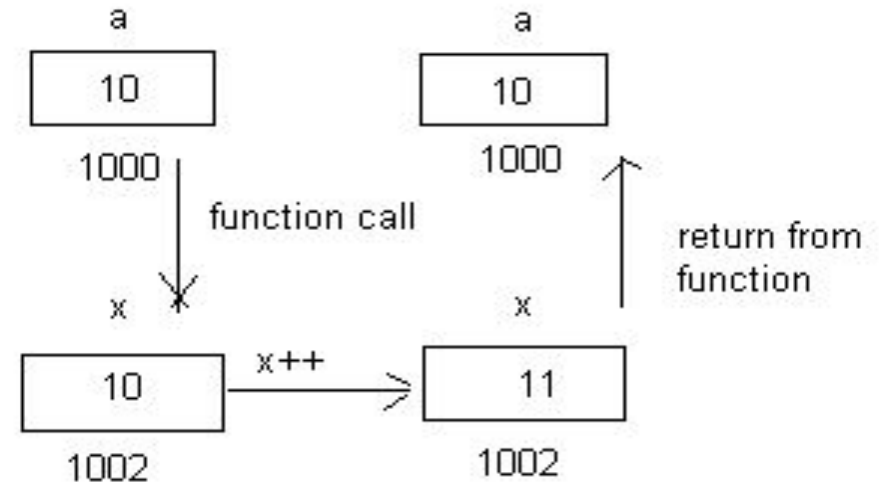
Calling a function by passing pointers as parameters (address of variables is passed instead of variables)

int a=1;

fun(&a);

void fun(int *x)

{

defn;

}

Any modification done to variable a will effect outside the function also

```
#include<stdio.h>
void main()
{
    int a=10;
    printf("%d",a);          a=10
    fun(a);
    printf("%d",a);          a=11
}
void fun(int x)
{
    printf("%d",x)           x=10
    x++;
    printf("%d",x);          x=11
}
```
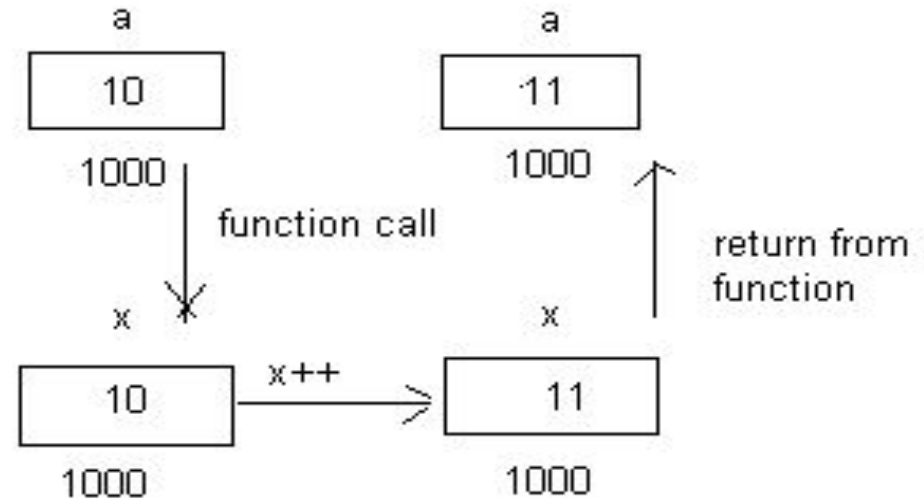
- Call by value => copying value of variable in another variable. So any change made in the copy will not affect the original location.

- Call by reference => Creating link for the parameter to the original location. Since the address is the same, changes to the parameter will refer to original location and the value will be overwritten.

Get familiar with C at

https://www.w3resource.com/c-programming-exercises/

A couple of suggestions:

- Basic Declarations and expressions

    1,5,10,28

- Input Output

    7

- Conditional Statement

    3

- Array

    2,3

Try to compile the programs on your PC!

Questions? Feedback? Suggestions?