Intro and Logistics

ENEE 457

Computer Systems Security

Fall 2021

Dana Dachman-Soled

- Normally, we care about **correctness**
  - Does software achieve desired behavior?

- Security is a kind of correctness
  - Does software prevent **undesired** behavior?

The key difference is the adversary!

What are undesired behaviors?

- Reveals info that users want to hide
  - Corporate secrets, private data, PII
  - *Privacy/Confidentiality*
- Modifies info or functionality
  - Destroy records, change data mid-processing, install unwanted software
  - *Integrity*
- Deny access to data or service
  - Crash website, DoS,
  - *Fairness*

Why are attacks so common?

- Systems are complex, people are limited
- Many attacks exploit a ***vulnerability***
  - A ***software defect*** that can be manipulated to yield an undesired behavior
- Software defects come from:
  - Flaws in design
  - Bugs in implementation

Why are attacks so common?

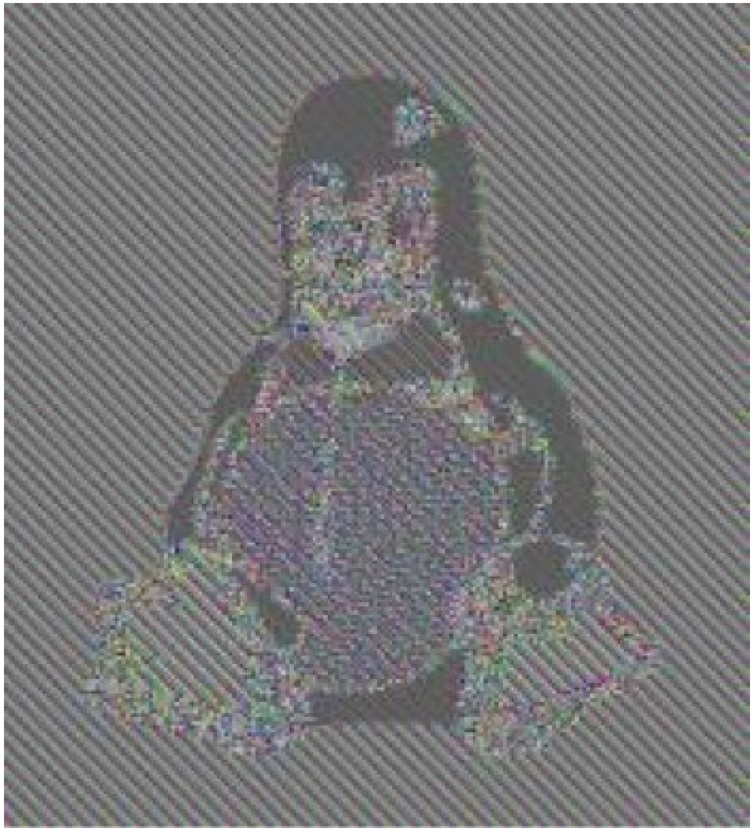
- Normal users avoid bugs
- Adversaries look for them to exploit

Why are attacks so common?

- Because it’s profitable
  - (Or attackers think it is)
- Because complex systems are only as strong as their weakest link

Steps toward more security….

- Eliminate bugs or design flaws, or make them harder to exploit
    - – Think like an attacker!
- Deeply understand systems we build
- Be mindful of user-controlled inputs



Today's agenda

- What is security
- Logistics
- C Refresher (Pointers, Memory Allocation)
- Case Study: Heartbleed Attack
- Course Survey

People

- Me: Dana Dachman-Soled (danadach@umd.edu)

- TAs:
  - Pranjal Atrey (patrey@terpmail.umd.edu)
  - Zahra Zarejousheghani (zzare@terpmail.umd.edu)

Lecture and Office Hours

- Assuming in-person lecture for now.
  - Mon/Wed 11am-12:15pm EGR 0108
  - Pre-recorded Lectures also posted
- Posted videos if instructor needs to miss a class
- Instructor Office Hours:
  - Times: Mon 1-2pm, Fri 10-11am
  - Location: Iribe 5238
- TA Office Hours:
  - Times: Zahra: Wed 12:30-1:30pm, Pranjal: Thurs 12-1pm
  - Location TBA

Resources

- Make sure to regularly check the class website:
  - http://www.ece.umd.edu/~danadach/Security_Fall_21/
  - Announcements, assignments, lecture notes, readings
- We will be using the Canvas page for the class
  - Recorded lectures
  - Announcements, grades, Project/HW submission, solutions
- Exams
  - Midterm exam will be held during class time
  - Final exam will be held at regularly scheduled time
- We will also use Piazza
  - Discussion on class material, questions
  - You should have received an email invite

Reading

- No required textbook
- Recommended: textbooks, outside resources
  - Listed on website and syllabus
  - Share your recommendations on Piazza

Prerequisite knowledge

- Reasonably proficient in C and Unix
  - Refresher on C pointers/memory allocation today
  - Experience in programming an end-to-end application
- Creative and resourceful
- No prior knowledge in networking, crypto

Grading

- Projects: 35%
  - Projects: 7%, 7%, 7%, 7%, 7%
- Homeworks: 6%
  - Will have either 2 or 3, if 3 then lowest grade dropped
- In-Class Labs 9%
  - Will have either 3 or 4, if 4 then lowest grade dropped
  - Tutorials will replace the pre-recorded lectures
  - Expect to take about 1 hour 15 min to complete
- Midterm: 25%
  - Tentative date: Wednesday October 20
- Final: 25%
  - Friday, December 17, 8-10am

Ethics and legality

- You will learn about, implement attacks
- ***Do not use them without explicit written consent from everyone involved!***
  - Make sure you know who is involved
- If you want to try something, tell me and I will try to help set up a test environment
- Don't violate: Ethics, UMD policies, state and national laws

Read the syllabus

- In general, no late projects/homework accepted.
  - The instructor may allow late homework submission under extenuating circumstances.
  - In this case documentation such as a doctor's note will be requested.
- Excused absences for exams
- Contesting project/exam grade
- Academic integrity
- Extra Credit opportunities

Action Items

- If you registered late, you may not have gotten an invite to Piazza
  - Please email me to let me know
- Check out the course webpage
  - https://user.eng.umd.edu/~danadach/Security_Fall_21/
- Project 1 is already posted on the course webpage
  - We haven't yet covered the background needed to complete it (mostly Lecture 2, some Lecture 3)
  - You can get started by installing Virtual Box and setting up the Virtual Machine

What’s in this course?

- Software and Web security
- Crypto
- Network security
- Special Topics (Bitcoin, Side-Channels, and more)

Software security

Memory safety

Malware

Web security

Static analysis

Design principles

What’s in this course?

- Software and Web security
- Crypto
- Network security
- Special Topics (Bitcoin, Side-Channels, and more)

Applied crypto

- What it is (medium-high level)
- How to use it responsibly

Black-box approach

Designing protocols that use crypto

Authentication

Public Key/Symmetric Key

What’s in this course?

- Software and Web Security
- Crypto
- Network security
- Special Topics (Bitcoin, Side-Channels, and more)

Network security

- How to build secure networked systems

Attacks on TCP, DNS, Packet Sniffing

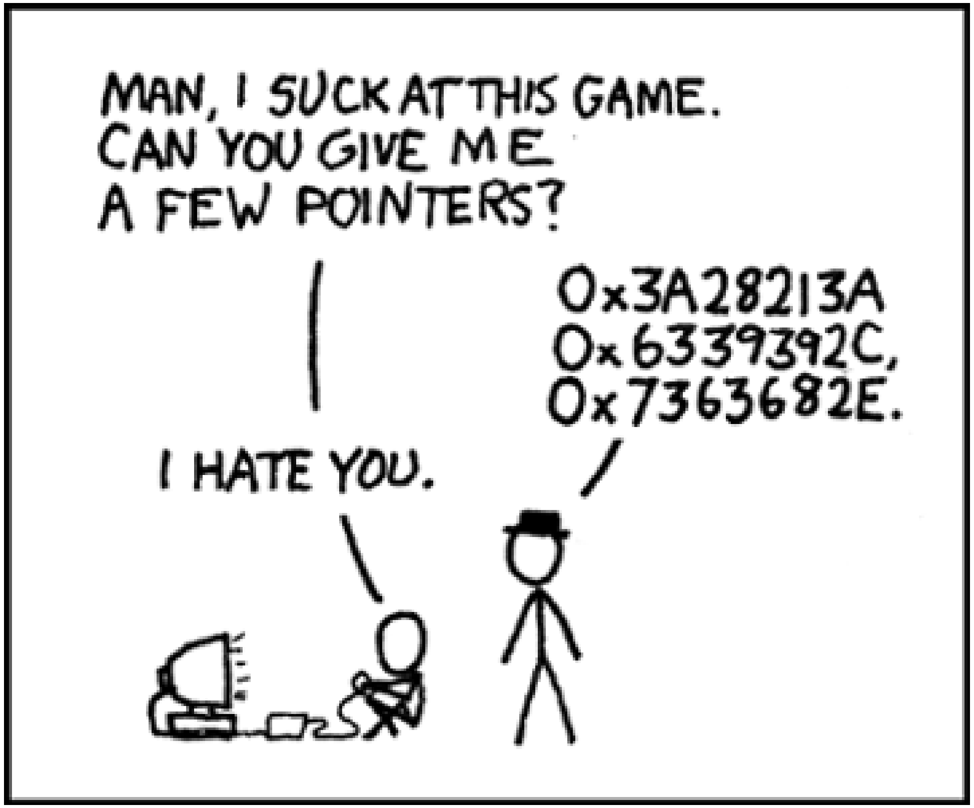
Anonymity

What's in this course?

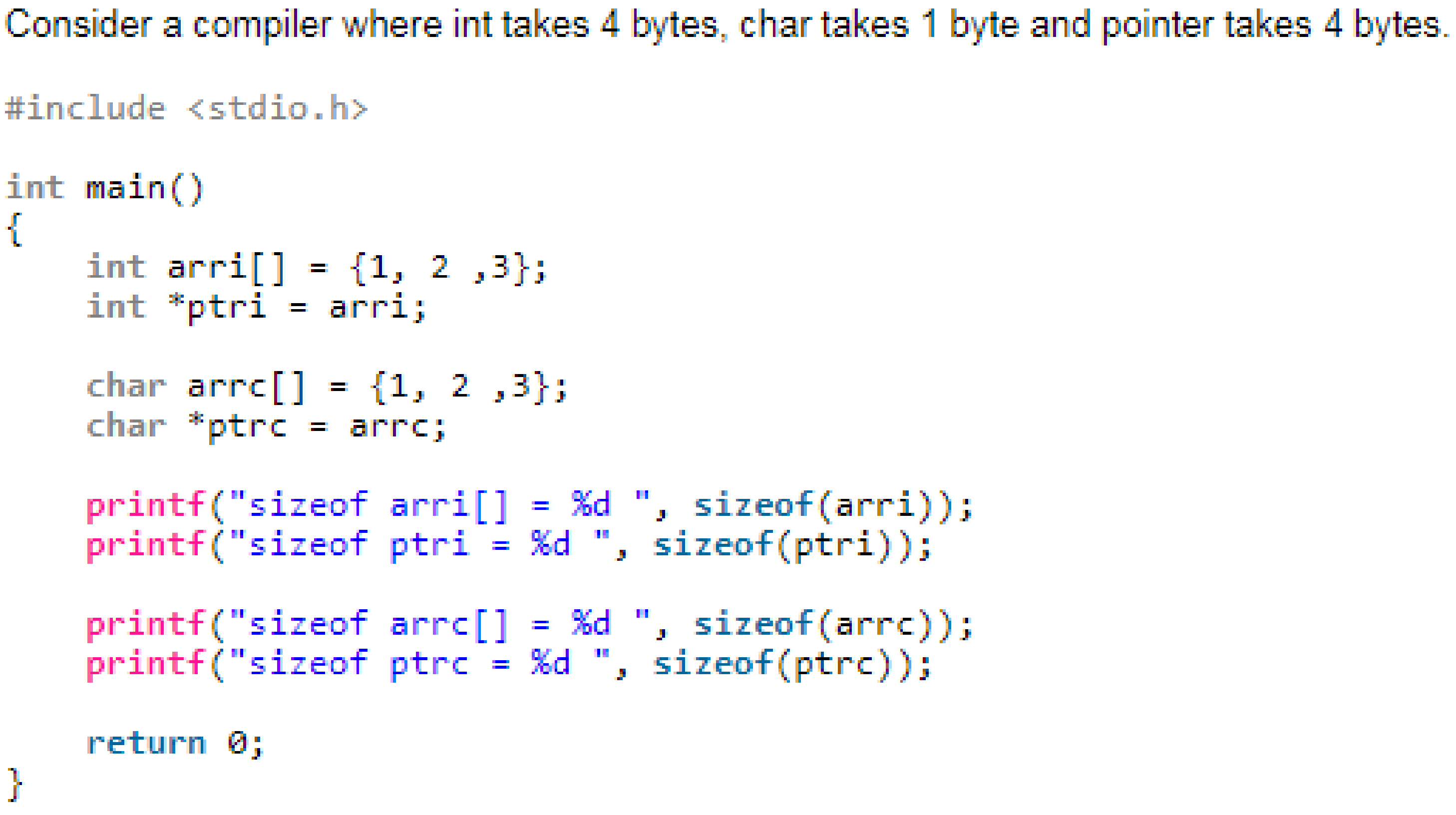
- Software and Web security
- Crypto
- Network security
- Special Topics (will include some or all of):
  - Bitcoin/Blockchain
  - Adversarial Machine Learning
  - Password Hashing
  - Side-Channel Attacks
  - Differential Privacy

First Topic: Buffer Overflows

Review:
Pointers and Memory Allocation in C



Review:
Pointers and Memory Allocation in C

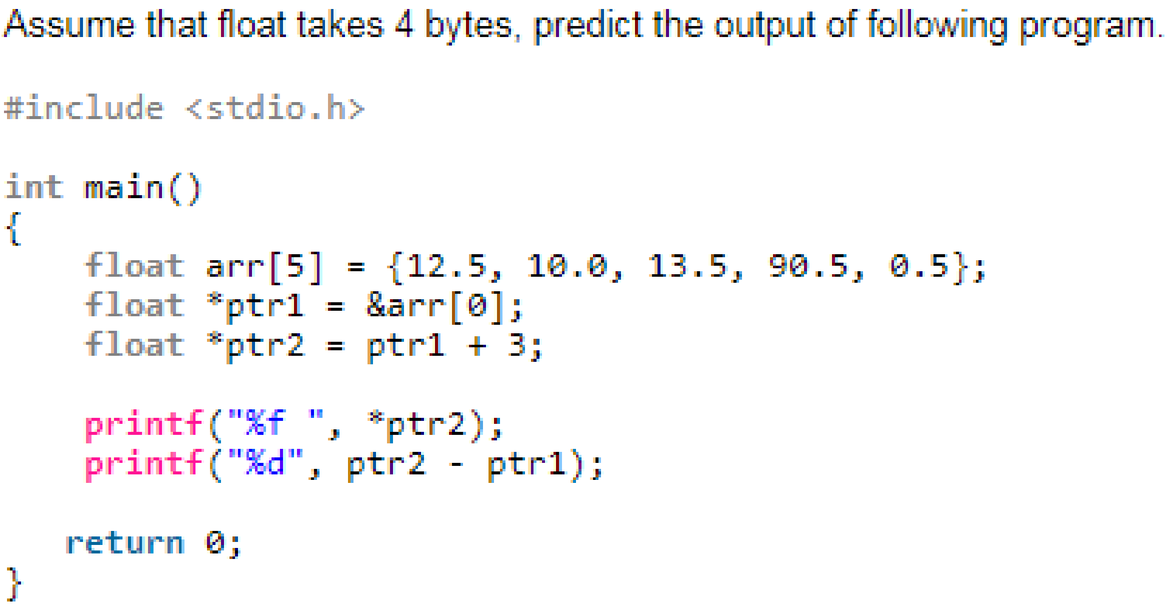


Consider a compiler where int takes 4 bytes, char takes 1 byte and pointer takes 4 bytes.

```
#include <stdio.h>

int main()
{
    int arri[] = {1, 2 ,3};
    int *ptri = arri;

    char arrc[] = {1, 2 ,3};
    char *ptrc = arrc;

    printf("sizeof arri[] = %d ", sizeof(arri));
    printf("sizeof ptri = %d ", sizeof(ptri));

    printf("sizeof arrc[] = %d ", sizeof(arrc));
    printf("sizeof ptrc = %d ", sizeof(ptrc));

    return 0;
}
```

Code

Review:
Pointers and Memory Allocation in C

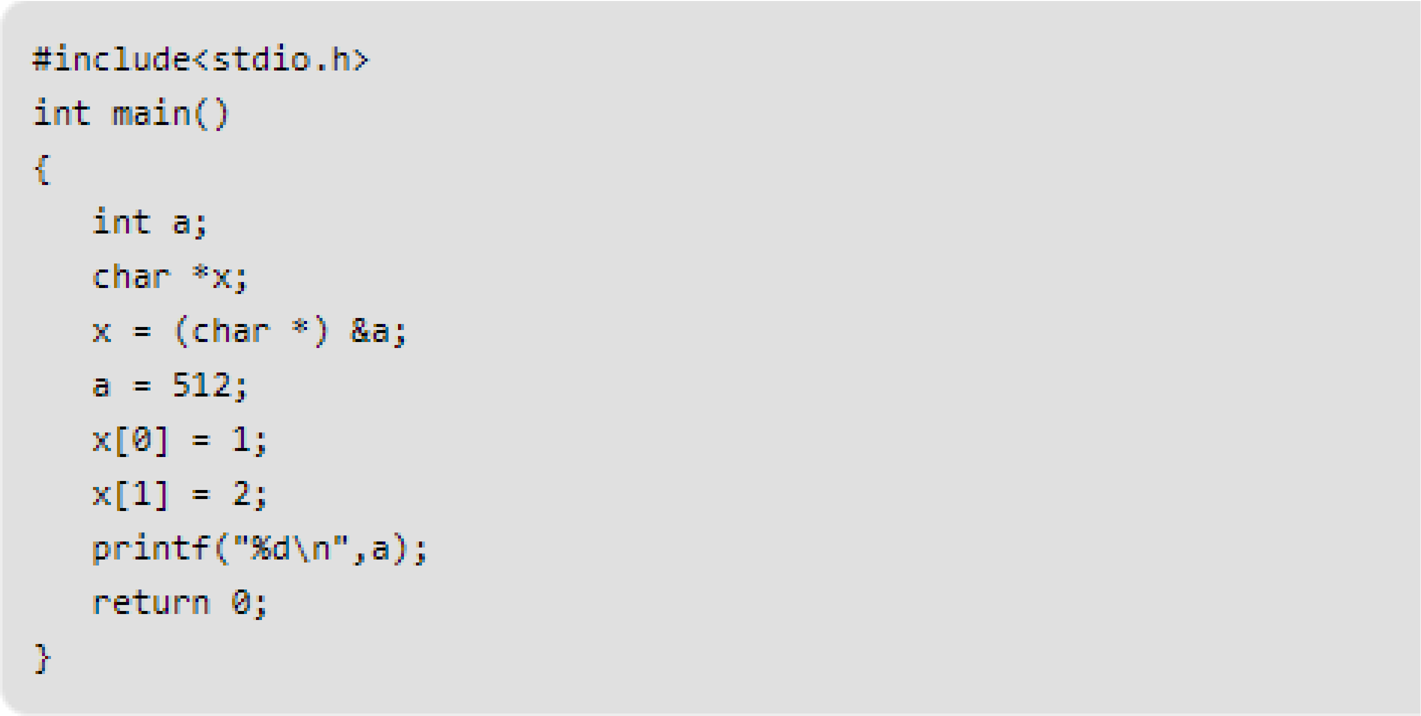


Assume that float takes 4 bytes, predict the output of following program.

```
#include <stdio.h>

int main()
{
    float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
    float *ptr1 = &arr[0];
    float *ptr2 = ptr1 + 3;

    printf("%f ", *ptr2);
    printf("%d", ptr2 - ptr1);

   return 0;
}
```

Code

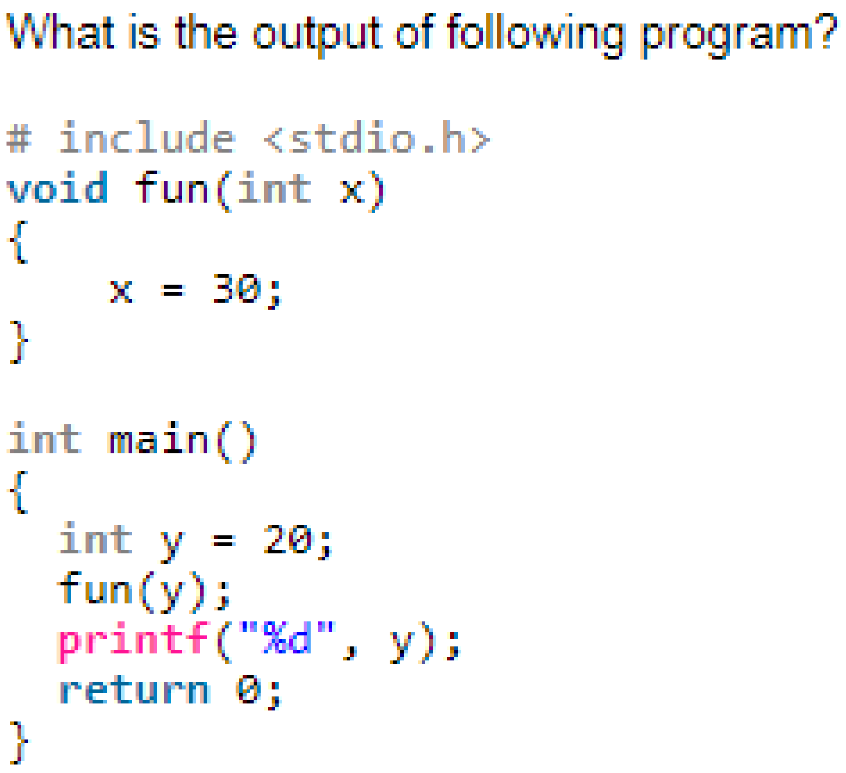
Review:
Pointers and Memory Allocation in C



```
#include<stdio.h>
int main()
{
   int a;
   char *x;
   x = (char *) &a;
   a = 512;
   x[0] = 1;
   x[1] = 2;
   printf("%d\n",a);
   return 0;
}
```

What is the output? Assume *little-endian* processor.
The **least significant** byte (the "little end") of the data is placed at the byte with the lowest address. The rest of the data is placed in order in the next three bytes in memory. Code

Review:
Pointers and Memory Allocation in C

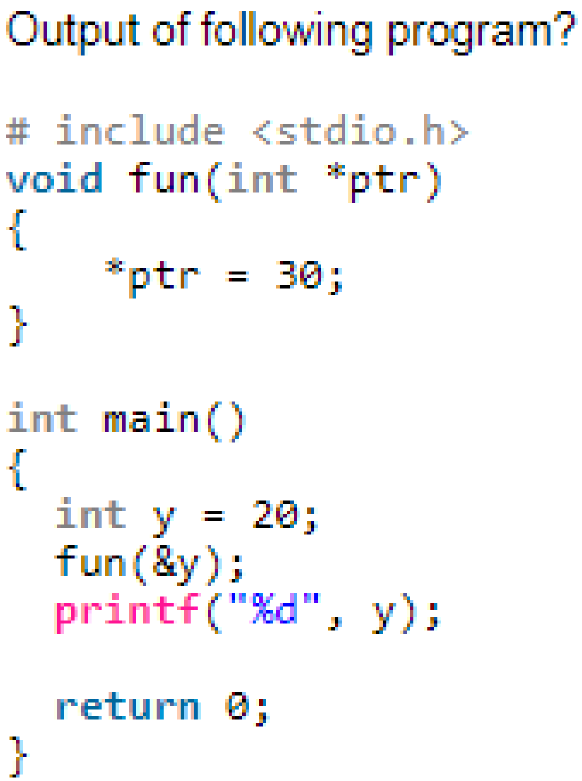


What is the output of following program?

```
# include <stdio.h>
void fun(int x)
{
        x = 30;
}

int main()
{
    int y = 20;
    fun(y);
    printf("%d", y);
    return 0;
}
```

Code

Review:
Pointers and Memory Allocation in C

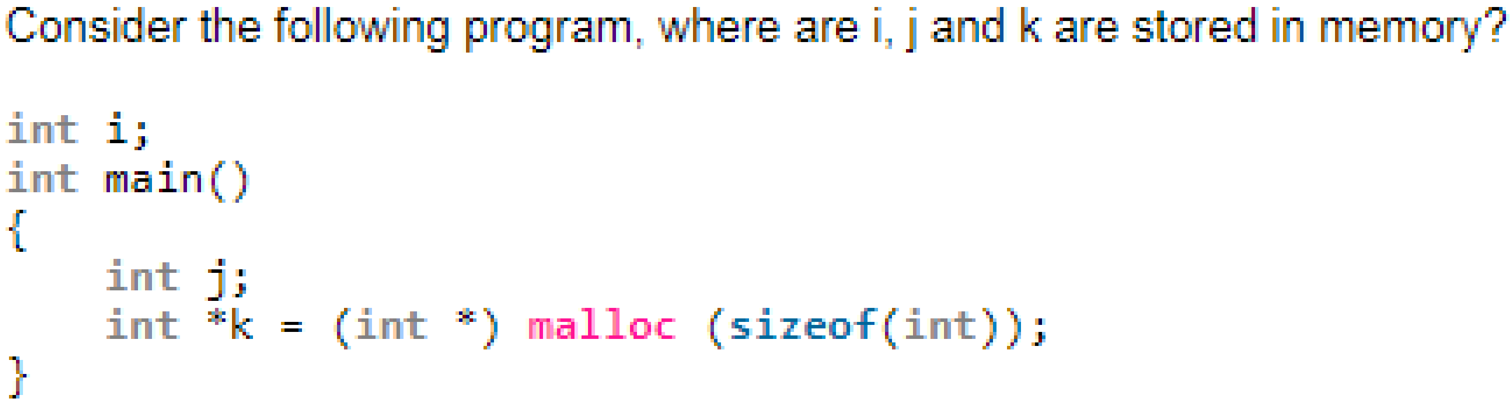


Output of following program?

```
# include <stdio.h>
void fun(int *ptr)
{
    *ptr = 30;
}

int main()
{
  int y = 20;
  fun(&y);
  printf("%d", y);

  return 0;
}
```

Code

Review:
Pointers and Memory Allocation in C

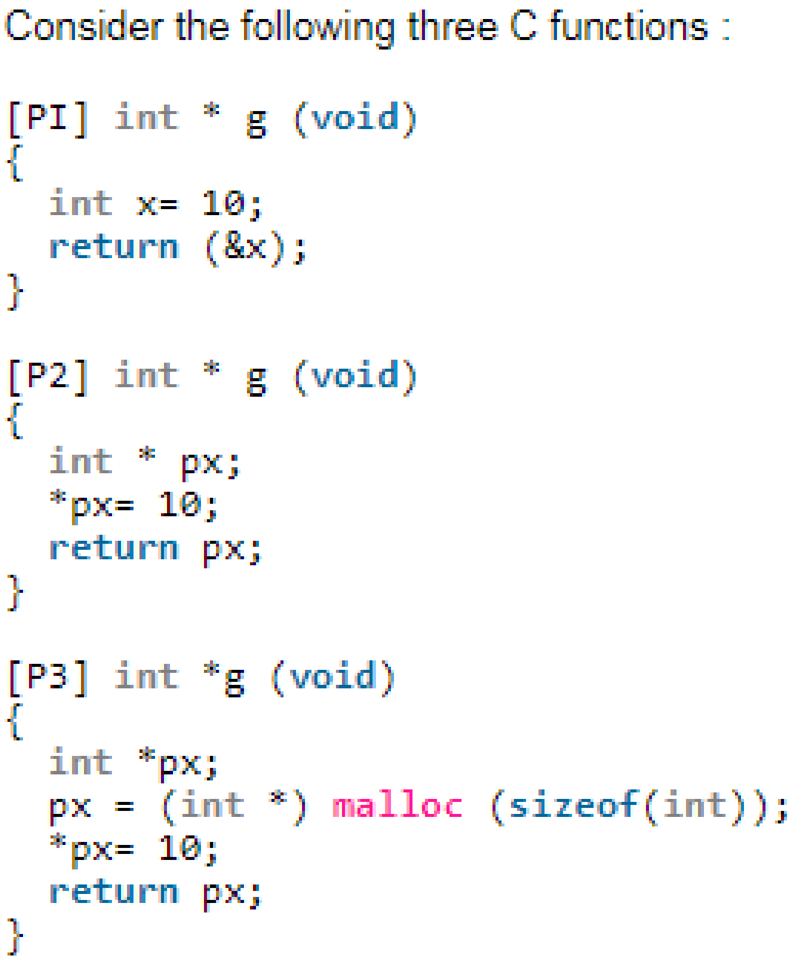


Consider the following program, where are i, j and k are stored in memory?

```
int i;
int main()
{
    int j;
    int *k = (int *) malloc (sizeof(int));
}
```

Code

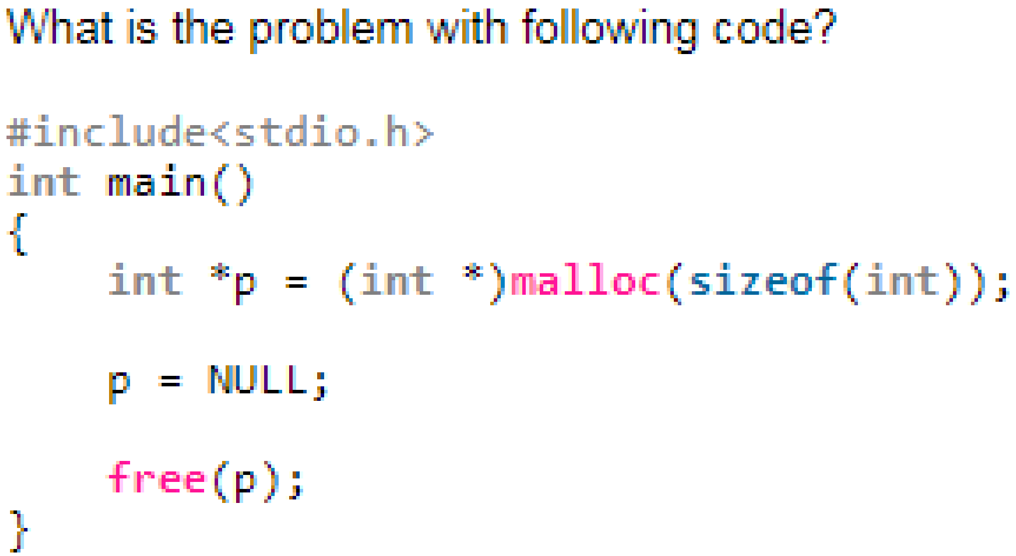
Review:
Pointers and Memory Allocation in C



Consider the following three C functions :

```
[PI] int * g (void)
{
  int x= 10;
  return (&x);
}

[P2] int * g (void)
{
  int * px;
  *px= 10;
  return px;
}

[P3] int *g (void)
{
  int *px;
  px = (int *) malloc (sizeof(int));
  *px= 10;
  return px;
}
```

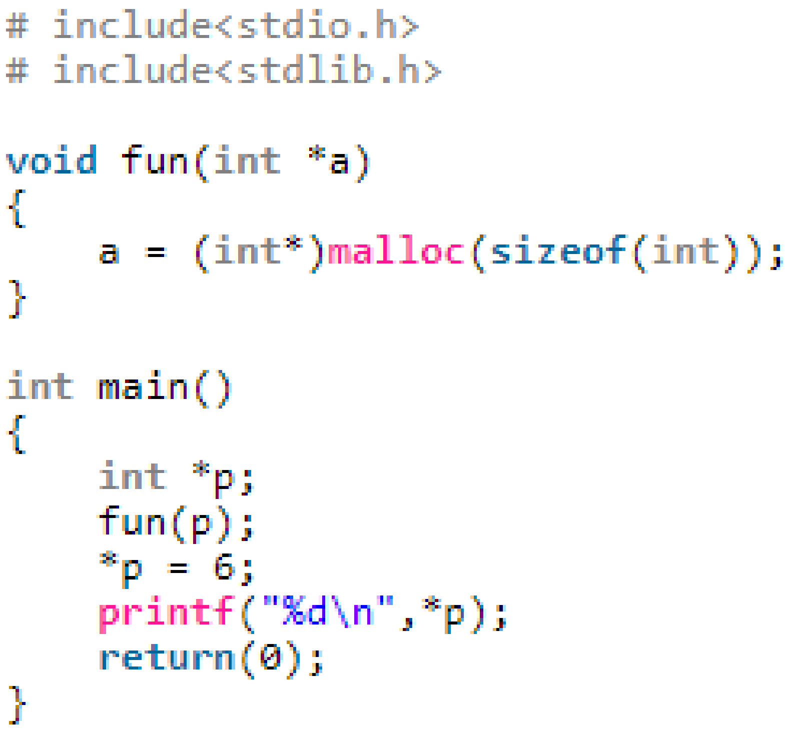
Review:
Pointers and Memory Allocation in C



What is the problem with following code?

```
#include<stdio.h>
int main()
{
    int *p = (int *)malloc(sizeof(int));

    p = NULL;

    free(p);
}
```

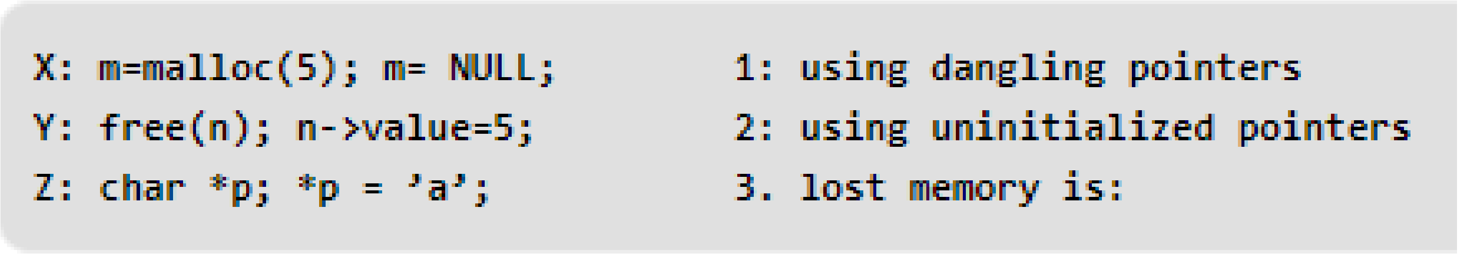
Review:
Pointers and Memory Allocation in C



```
# include<stdio.h>
# include<stdlib.h>

void fun(int *a)
{
    a = (int*)malloc(sizeof(int));
}

int main()
{
    int *p;
    fun(p);
    *p = 6;
    printf("%d\n",*p);
    return(0);
}
```

Code

Review:
Pointers and Memory Allocation in C



```
X: m=malloc(5); m= NULL;          1: using dangling pointers
Y: free(n); n->value=5;           2: using uninitialized pointers
Z: char *p; *p = 'a';             3. lost memory is:
```

C Refresher Topics:

Some Refresher Topics in C

This is a list of topics relevant for the Memory Safety/Buffer Overflows unit, which is our first unit in the course. For the Build-It-Break-It Project (the final 2 projects of the course), overall comfort and fluency in programming will be most useful, as opposed to knowledge of any single topic.

Topics on Pointers and Memory Allocation:

- sizes of datatypes (e.g. assuming 32-bit addressing so pointers/ integers are 4 bytes)
- pointer arithmetic (effects of incrementing a pointer depend on the datatype)
- dereferencing a pointer using * and getting the address of a variable using &
- "pass by value" and implications
- local variables are not saved when a function call returns and implications
- correct usage of "malloc“, correct usage of "free", setting pointers to NULL and issues with memory leaks/dangling pointers

Additional Topics:

- c strings (NULL terminated)
- scanf, strcpy, strncpy, and issues they can cause with buffer overflow
- integer overflow
- usage of printf with/without format specifiers
- basic usage of c structs
- basic usage of function pointers

SEED Lab Setup:

Home: https://seedsecuritylabs.org/index.html
Lab Setup: https://seedsecuritylabs.org/labsetup.html

Our first project (note that not all tasks are required. See course webpage):

https://seedsecuritylabs.org/Labs_20.04/Files/Buffer_Overflow_Setuid/Buffer_Overflow_Setuid.pdf

https://seedsecuritylabs.org/Labs_20.04/Files/Return_to_Libc/Return_to_Libc.pdf