Introduction

IE 5995: IoT and Edge AI Programming
Dr. Yanchao Liu
Industrial & Systems Engineering
Wayne State University
Agenda

• Course organization and syllabus
• Machine learning (ML) and Artificial Intelligence (AI)
• Internet of Things (IoT)
• Edge AI
• Applications
• Development workflow
• C programming language refresher
Yanchao Liu
B.S. in IE, Huazhong Univ. of Sci. & Tech., 2006
M.S. in IE, Univ. of Arkansas, 2008
Ph.D. in ISE, UW-Madison, 2014
Assistant Prof. at WSU since 2017.

• Data Scientist, Sears Holdings Corporation (SHC)
• Manager of Advanced Analytics, SHC
• Director of Brand Marketing Analytics, Catalina Marketing Corporation

Married, two children, live in Troy:
• Amy is in kindergarten
• Anthony is in pre-school

yanchaoliu@wayne.edu; 313-577-3301; MEB 2169

Your turn: upload a short bio with picture by Jan 21
Course plan

• Explore IoT and Edge AI technologies and applications

• Understand the basic workflow of building and deploying ML and AI

• Interact with device hardware via programming in C and MicroPython

• Experience several popular MCU families suitable for IoT and Edge AI development

• Explore the IoT data processing life cycle, including the collection, transmission, storage and analysis of data
Prerequisites, grading and learning approach

• Inclination toward computer programming
• Engineering mindset
• Inquisitive about the physical world

• Grading scheme: Homework 40%, Participation 10%, Project 50%
  • Homework: hybrid grading – show your work in class
  • Participation: ask questions, answer questions, be active
  • Project: non-trivial implementation of something useful by applying knowledge including and beyond what’s learned in class

• Learning approach:
  • Type up your own code, and make it work on your device
  • Learn from sample code, assimilate then modify, integrate or extend
  • Be ready to show your work
  • Read manuals and product specification documents
Textbook & Useful References

https://micropython.org/
https://www.adafruit.com/
https://www.arduino.cc/
https://www.nordicsemi.com/
https://www.sparkfun.com/
https://www.tensorflow.org/lite/examples
Materials

Windows PC with at least 20 GB of hard drive space for software installation
Machine learning based vs Rule-based decisions

• Task: control a process by determining the control input value

• Rule-based approach:
  • Figure out the relation between input and outcome, by knowledge or experience
  • Implement if...then...rules into a computer program, i.e., if certain conditions are met, set the control value to $X_1$; if another condition is met, set it to $X_2$, etc.

• Machine learning based approach:
  • Feed the ML algorithm lots of prior data: (input, outcome), and let it figure out the pattern / relation, the insight is encoded into a "model"
  • Implement the model into a computer program

  - The two approaches only differ in how the content of the Program is generated.
  - From implementation perspective, the input -> function -> output paradigm is the same.
Neural network model

Rule-based model
Why use machine learning?

• Fundamental assumptions:
  • Labeled data is abundant and cheap
  • Data storage is cheap
  • Computing power is cheap
  • Data transmission is cheap
  • Predicting correctly is more important than understanding why

• Advantages:
  • Engineers do not need to be smart
  • When rules are hard to figure out
  • Insight generation program is shorter and easier to maintain
  • Hype – people buy it

• Disadvantages:
  • Competitive edge relies on data dominance
  • Model size and computing burden in inference stage
Deep Learning

• Deep learning is a special kind of machine learning
  • Uses artificial neural network (ANN) to organize relationships among data
  • The model training process is of no difference from that of other ML algorithms

• Workflow
  • Decide on a goal
  • Collect labeled data sets
  • Choose / design a model architecture
  • Train the model
  • Write the model into a computer program
  • Run inference
Internet of Things (IoT)

- **Enabling technologies**
  - Low-cost low-power processor chips with integrated RF module
  - Increased wireless communication bandwidth (cellular, WIFI, Bluetooth)
  - Adoption of IPv6
  - New materials and sensors

- **Applications**
  - Smart Home (lock, lightbulb, speaker, Amazon’s Echo, Google Home)
  - Wearable devices (fitness, location tracking, health monitoring)
  - Surveillance and security, fleet telematics, emergency services
  - Industrial automation, sensor networks, city infrastructure

- **Challenges**
  - Privacy, security
Industrial Applications

Telehealth Remote Monitoring
Immersive Media
Vehicle Event Recording
Cargo Management

Defect detection, quality control, workspace safety, AI-powered inspection
Smart Home Applications

- Intrusion detection
- Surveillance
- Remote control
- Event logging
- Leakage detection
- Programmable décor
- Lighting, music, alerts, security
Smart Health Applications

- Wearable gadgets
- Ambient assisted living
- Adverse drug reaction detection
- Medication management
- Wheelchair add-ons
- Sleep quality evaluation
Smart City Infrastructure

- Traffic & parking management
- Waste management
- Adaptive lighting
- Urban maintenance
- Air mobility and drone delivery
- Community safety

Research by Zion Market Research reveals that the IoT market for Smart Cities will grow by 22.6% to $330.1 billion in 2025.
Edge AI

• Artificial Intelligence deployed on-device
  • Computing happens where result is needed
  • Data is consumed where it is generated - on premise/device/vehicle
  • Low latency
  • Privacy
    • Audio, image, video, biometrics
  • Robustness - core functions do not rely on network connection
  • Example applications
    • Electric shock detection - wearable device
    • Fall detection - parachute trigger for drones
    • Smart device - learning on the fly
      • Coffee machine
      • Refrigerator
      • Wheelchair
      • Electric / water meter
      • Toys
IoT Data Management

- Edge computing with Cloud support
- Over-the-Air (OTA) programming / update / administration
- Real-time database on the Cloud

- Cellular LTE / 5G
- Wired connection via router
- VPN
- Cloud platform (GCP, AWS)

- WiFi
- Bluetooth
- IEEE 802.11p
IoT Product Design Process

• Conceptualization
  • Purpose of product, who need it, what benefit, how much a customer would pay

• Requirement
  • Functions, features, power, comm, UI, precision, size, weight, cost limit
  • Detailed specifications, conditions, compliance, BOM, manufacturing, environment

• Prototyping
  • Can be physical or virtual, test hypothesis, verify customer needs

• Architecting
  • Platform / OS / language selection, components licensing

• Coding / Debugging

• Production

• Maintenance
  • Updates, bug fixes
Reverse Engineering a Product

Kenmore Alfie

Winbond 25032JVS10

CC3200R1 M2
Warm up with programming in THE C PROGRAMMING LANGUAGE
Set up the exercise environment

• There are several options:
  • Use the virtual machine: vlab.vdi.wayne.edu
  • Install Ubuntu Linux on Windows PC using VirtualBox
  • Install Microsoft Visual Studio
  • Online coding / compiling tools, such as
    https://www.onlinegdb.com/online_c_compiler
A C program, whatever its size, consists of functions and variables.
- A function contains statements that specify the computing operation to be done
- Variables store values used during computation.

You are at liberty to give functions whatever names you like
"main" is special: the program begins executing at the beginning of main

- The parentheses after the function name surround the argument list.
- In this example, main expects no arguments
- The statements of a function are enclosed in braces {}
- A statement is terminated by a semicolon ;
- Indentation and whitespace does not matter to C compiler
- Compiler directives should not end with ; (unless you want to define an alias for something that ends with ;)
- The C compiler treats code case-sensitive
Variables and Arithmetic Expressions

```c
#include <stdio.h>
/* print Fahrenheit-Celsius table */
for fahr = 0, 20, ..., 300 */
main() {
  int fahr, celsius;
  int lower, upper, step;

  lower = 0; /* lower limit of temperature table */
  upper = 300; /* upper limit */
  step = 20; /* step size */

  fahr = lower;
  while (fahr <= upper) {
    celsius = 5 * (fahr-32) / 9;
    printf("%3d %6d\n", fahr, celsius);
    fahr = fahr + step;
  } // end while loop
}
```

- Any characters between /* and */ are ignored by the compiler
- Any characters following // will be ignored till the end of the line (this is C++ syntax but works with C)
- All variables must be declared before they are used
- Integer division truncates the fractional part of the result

Exercises:
1. Change int to float, and change the printf statement to
   ```c
   printf("%3.0f %6.1f\n", fahr, celsius);
   ```
2. Print a heading above the table
3. Use for statement instead of while statement
4. Define symbolic constants using
   ```c
   #define NAME replacement_text
   ```
Functions

```c
#include <stdio.h>

int power(int m, int n);

int main() {
    int i;
    for (i = 0; i < 10; ++i)
        printf("%d %d %d\n", i, power(2, i), power(-3, i));
    return 0;
}

int power(int base, int n) {
    int i, p;
    p = 1;
    for (i = 1; i <= n; ++i)
        p = p * base;
    return p;
}
```

- Function declaration, or prototype, must agree with the definition and use of the function
- Parameter types must agree between function prototype and function definition, parameter names need not (parameter names in the function prototype can be omitted)
- The names used by a function as its parameters are local to the function, and are not visible to any other function
- Variables declared with in a {} block is local to the block
- As a convention, we use parameter for a variable named in the parenthesized list in a function definition, and argument for the value used in a call of the function.
- In C, all function arguments are passed “by value”
- The called function cannot directly alter a variable in the calling function, it can only alter its private, temporary copy.
Scope of a variable

```c
#include <stdio.h>

#define MAXLINE 30

char buf[MAXLINE];

int setBuf(int val){
    extern char buf[];  // can be omitted
    return sprintf(buf, "The value is %d\n", val);
}

void printBuf(int n){
    for(int i = 0; i < n; i++){
        printf("%c", buf[i]);
    }
    printf("\n");
}

int main(){
    printBuf(setBuf(100));
    return 0;
}
```

- Array subscripts always start at zero in C.
- Variables declared within a function are private or local to the function, they come and go with function invocation, and do not retain their value from one call to the next.
- An external variable must be `defined`, exactly once, outside of any function, and must be `declared` with “extern” keyword in each function that wants to access it.
- The extern declaration can be omitted if the definition of an external variable occurs before its use in a particular function.
- `Definition` creates a variable and sets aside storage for it; `declaration` states the nature of the variable, but no storage is allocated.
Constants

- A variable can be initialized in its declaration
- External and/or static variables are initialized to zero by default
- Automatic variables without an explicit initializer have undefined values
- An integer constant like 1234 is an int.
- A long constant is written with a terminal l or L, e.g., 123456789L
- Unsigned constants are written with a terminal u or U, e.g., 123u
- Unsigned long is indicated by the suffix ul or UL, e.g., 123456789UL
- A leading 0 (zero) on an integer constant means octal
- A leading 0x or 0X means hexadecimal
  - Example: decimal 31 can be written as 037 in octal and 0x1f in hex
- Example: 0xFUL is an unsigned long constant with value 15 in decimal
- A character constant is an integer, written as one character within single quotes, such as ‘x’
- The value of a character constant is the numeric value of the character in the machine’s character set, e.g., ‘0’ has the value 48 in ASCII character set
- A string constant, or a string literal, is a sequence of zero or more characters surrounded by double quotes, as in “I am a string”
- String constants can be concatenated at compile time, so “hello,” “world” is equivalent to “hello, world”
- A string constant is an array of characters. The physical storage required for a string is one more than the number of characters written between the quotes, because a string always has a null character ‘\0’ at the end.

```c
#include <stdio.h>
#include <string.h>

int inc(void);
void main(){
    int a = 037;
    unsigned long b = 0x1fUL;
    const char msg[] = "warning: ";
    const double e = 2.71828182845905;
    printf("%ld\n", 1234567L);
    printf("%d %o %x %c \n", a, a, a, a);
    printf("%ld %lo %lx %c \n", b, b, b, (int)b);
    printf("%ld\n", strlen(msg));
    printf("count" " values:\n");
    printf("%d \n", inc());
    printf("%d \n", inc());
}

int inc(){
    static int count = 0;
    count++;
    return count;
}
```

- ++n increments (add 1 to) n before its value is used
- n++ increments n after its value has been used
- A static variable preserves its value in its previous scope and is not initialized again in the new scope
Handling text streams

```
#include <stdio.h>
/* copy input to output */
main(){
    int c;
    c = getchar();
    while(c != EOF){
        putchar(c);
        c = getchar();
    }
}
```

```
#include <stdio.h>
/* count characters in input */
main(){
    long nc;
    while(getchar() != EOF)
        ++nc;
    printf("%ld\n", nc);
}
```

- `getchar()` reads and returns the next character from the standard input
- `putchar(c)` prints the content of the integer variable `c` as a character
- `EOF` is an integer defined in `<stdio.h>`, for “end of file”; it can be typed as Ctrl-D keystroke in a terminal after a newline character (i.e., Enter)

- In C, any assignment, such as `c = getchar()`, is an expression and has a value, which is the value of the left-hand side after the assignment

```
#include <stdio.h>
/* copy input to output */
main(){
    int c;
    while((c = getchar()) != EOF)
        putchar(c);
}
```
Word Counting

#include <stdio.h>

#define IN 1
#define OUT 0

/* counts lines, words and characters in input */

main()
{
    int c, nl, nw, nc, state;
    state = OUT;
    nl = nw = nc = 0;
    while ((c = getchar()) != EOF) {
        ++nc;
        if (c == '\n')
            ++nl;
        if (c == ' ' || c == '\n' || c == '\t')
            state = OUT;
        else if (state == OUT) {
            state = IN;
            ++nw;
        }
    }
    printf("%d %d %d\n", nl, nw, nc);
}
Homework (due 1/21 before class)

• Write a program to print out the binary representation of an unsigned integer constant
• Write a program to print the histogram of the lengths of words in its input
• Write a function \texttt{reverse(s)} that reverses the character string \texttt{s}. Use it to write a program that reverses its input a line at a time.