Unit One Programming Language

1.1 Essential Reading: C Programming Language

1. The Fundamental Components of a Program

Data types, operators[1], and expressions are the most fundamental components of a program and serve as the foundation for learning any programming language. This section introduces the basic concepts of C programming, including data types, variables[2], constants[3], operators, and expressions[4], as well as methods for writing simple programs using basic input/output functions.

(1) In a C program, every variable, constant, and expression has a specific data type. The data type explicitly[5] or implicitly[6] defines the range of possible values that a variable or expression can take during program execution, as well as the operations allowed on those values. The primary data types in C can be classified into four categories: basic data types, derived data types[7], pointer types[8], and the void type[9].

(2) C provides a rich set of operators to perform complex expression evaluations. In general, unary operators[10] have higher precedence[11], while assignment operators[12] have lower precedence. Arithmetic[13] operators have higher precedence than relational[14] and logical operators. Most operators have left associativity[15], whereas unary, ternary[16], and assignment operators have right associativity.

(3) An expression is a combination of operators, constants, variables, and functions. Each expression has a value and a data type. The evaluation of expressions follows the order determined by operator precedence and associativity.

(4) C supports two methods of type conversion: implicit (automatic) and explicit (forced) conversion.

- **Implicit type conversion**: In mixed-type operations, the system automatically converts data from a smaller-sized type to a larger-sized type. When assigning values between different types, the right-hand side of the assignment is automatically converted to the type of the left-hand side.
- **Explicit type conversion**: Achieved using a cast operator to force a type conversion.

[1] operator: 操作符

[2] variable: 变量

[3] constant: 常量

[4] expression: 表达式

[5] explicitly: 明显地

[6] implicitly: 隐含地

[7] derived data type: 构造数据类型

[8] pointer type: 指针类型

[9] void type: 空类型

[10] unary operator: 单目运算符

[11] precedence: 优先级

[12] assignment operator: 赋值运算符

[13] arithmetic: 算术

[14] relational: 关系

[15] left associativity: 左结合性

[16] ternary: 三目

(5) Standard input/output refers to data entry via the keyboard and output via the display, with the computer as the central processing unit. C's standard input/output functions are included in the stdio.h library[17].

- Getchar() and putchar() are character I/O functions, handling one character at a time.
- Scanf() and printf() are formatted I/O functions, capable of reading and writing data in various types and formats.
- The format control string[18] is a crucial part of formatted I/O functions, ensuring correct data input and display.

[17] library: 库

[18] format control string: 格式控制字符串

2. The Three Basic Program Structures

The principles of structured programming mainly follow three key concepts: top-down design[19], stepwise refinement[20], and modularity[21]. In C, modularity is primarily achieved through functions. Practice has shown that applying the three basic program structures—sequential structure[22], selection structure[23](branching structure), and loop structure[24]—can effectively solve problems in structured programming.

[19] top-down design: 自顶向下设计

[20] stepwise refinement: 逐步细化

[21] modularity: 模块化

[22] sequential structure: 顺序结构

[23] selection structure: 选择结构

[24] loop structure: 循环结构

(1) Sequential Structure

Among the three basic program structures, the sequential structure is the simplest and most fundamental flow control structure, executing statements in a top-down order. A program module consists of Module 1 and Module 2, where Module 1 is executed first, followed by Module 2. The entire module has a single entry point (top) and a single exit point (bottom). Here, Module 1 and Module 2 can be a single statement or multiple statements (including complex structures).

In structured programming, the sequential structure is the most common flow control pattern. A complete C program can be viewed as a sequential structure composed of three parts: data input, data processing, and data output, executed in order.

For sequential structures, program execution strictly follows the written order of statements. No matter how complex a C program is or how many statements it contains, the overall execution remains sequential. A C program starts execution from the first statement in the main() function and proceeds until the last statement, making the entire body of main() a sequential structure.

(2) Selection Structure

As the name suggests, the selection structure chooses one path from multiple possible options based on a condition. In C, selection is implemented using if and switch statements.

First, a condition is evaluated.

- ◇ If the condition is true, Statement 1 is executed, and then the program exits the selection structure.
- ◇ If the condition is false, Statement 2 is executed, and the program exits afterward.

C provides two selection control statements:

- ◆ If statements (with three forms: if, if-else, and if-else if) for single-branch, double-branch, and multi-branch problems.
- ◆ Switch statements, primarily used for multi-branch selection problems.

Once a branch is executed, the program exits the selection structure and continues with the next statement. In switch statements, the break keyword is required to exit the structure explicitly.

(3) Loop Structure

The loop structure allows a part of the program to be executed repeatedly, making it essential in structured programming. Whether the loop continues depends on a termination condition[25]. Based on when the condition is checked, loops are categorized into:

- ◆ Pre-test loops[26] (while loops)—Check the condition before execution.
- ◆ Post-test loops (do-while loops)—Check the condition after execution.

C provides three loop control statements:

- ◆ While loop—Executes as long as the condition is true (best for loops with an unknown number of iterations).
- ◆ Do-while loop—Executes at least once before checking the condition.
- ◆ For loop—Typically used when the number of iterations is known.

The key feature of loops is that a block of code is repeatedly executed as long as a condition holds true.

Almost any complex problem can be solved using these three basic program structures. Proper application of these structures significantly improves code readability[27] and program efficiency[28].

[25] termination condition: 循环的终止条件

[26] pre-test loop: 当型循环

[27] code readability: 程序的可读性

[28] program efficiency: 程序效率

[29] decomposition: 分解

3. Functions in C Language

When solving complex problems, people often adopt a step-by-step decomposition[29] approach—dividing a large problem into smaller, more manageable sub-problems and solving them individually. Similarly, when designing complex applications, programmers typically break down the entire program into smaller, functionally independent modules,

implement them separately, and then integrate them like building blocks. This divide-and-conquer strategy in programming is known as modular programming[30]. In C, these modules are implemented as functions.

[30] modular programming: 模块化编程

Functions are the fundamental building blocks of a C program, which is why C is often referred to as a function-oriented[31] language. A C program consists of one or more functions, each performing a specific task. Every C program must have exactly one main() function, where execution begins. The main() function may call other functions before eventually returning control to itself and terminating the program.

[31] function-oriented: 面向函数的

From the perspective of function definition, C functions can be categorized into:

(1) Library Functions

Library functions are provided by the C compiler system. Users do not need to define them or declare their types explicitly—they only need to include the appropriate header file containing the function prototype[32]. Examples include frequently used functions like printf(), scanf(), getchar(), putchar(), and sqrt().

[32] prototype: 原型

To use a library function, the corresponding header file must be included using the #include preprocessor directive[33]. This allows the compiler to locate the function's object code and generate an executable. For example:

[33] preprocessor directive: 预处理器指令

- #include <math.h> for mathematical functions.
- #include <string.h> for string manipulation[34] functions.

[34] manipulation: 操纵

(2) User-defined Functions

User-defined functions are created by programmers to fulfill specific requirements. Unlike library functions, these must be fully defined in the program, and their return type must be declared in the calling function before they can be used.

From a functional standpoint, C functions can be divided into:

- Functions with return values—Return data to the caller.
- Functions without return values—Perform operations but do not return data (declared as void).

From a parameter-passing perspective, functions can be categorized as:

(1) Parameterless Functions

These functions do not accept any parameters in their definition, declaration, or call[35]. No data is passed between the calling and called functions. They are typically used to perform a fixed set of operations and may or may not return a value.

[35] call: 调用

(2) Parameterized Functions

Also known as functions with parameters, these functions include formal parameters[36] (parameters in definition) and require actual parameters[37] (arguments[38] passed during the call). When called, the calling function passes the actual parameter values to the formal parameters for use within the function.

[36] formal parameter: 形式参数
[37] actual parameter: 实际参数
[38] argument: 参数

Task 1 Read the passage above and then work in pairs to speak out the common expressions given in brackets.

1. This section introduces the basic concepts of C programming, including data types, _____(变量), _____(常量), _____(运算符), and _____(表达式), as well as methods for writing simple programs using basic input/output functions.

2. The primary data types in C can be classified into four categories: basic data types, _____(构造数据类型), _____(指针类型), and the _____(空类型).

3. Most operators have _____(左结合性), whereas _____(一元), _____(三元), and assignment operators have right associativity.

4. The principles of structured programming mainly follow three key concepts: _____(自顶向下), _____(逐步细化), and _____(模块化).

5. Practice has shown that applying the three basic program structures—_____(顺序结构), _____(选择结构)(branching structure), and _____(循环结构)—can effectively solve problems in structured programming.

6. The entire module has _____(一个入口点) (top) and _____(一个出口点) (bottom).

7. From the perspective of function definition, C functions can be categorized into: _____(库函数)and _____(用户定义函数).

8. To use a library function, the corresponding header file must be included using the _____(#include 预处理器指令).

Task 2 Read the sentences taken from the passage above and work in groups to translate them into Chinese.

1. The data type explicitly or implicitly defines the range of possible values that a variable or expression can take during program execution, as well as the operations allowed on those values.

_____

_____

2. In general, unary operators have higher precedence, while assignment operators have lower precedence. Arithmetic operators have higher precedence than relational and logical operators.

_____

_____

3. The evaluation of expressions follows the order determined by operator precedence and associativity.

4. In mixed-type operations, the system automatically converts data from a smaller-sized type to a larger-sized type.

5. C's standard input/output functions are included in the stdio.h library.

6. A C program starts execution from the first statement in the main() function and proceeds until the last statement, making the entire body of main() a sequential structure.

7. Once a branch is executed, the program exits the selection structure and continues with the next statement. In switch statements, the break keyword is required to exit the structure explicitly.

8. The key feature of loops is that a block of code is repeatedly executed as long as a condition holds true.

1.2 Advanced Reading: Code Similarity Detection

1. Research Prospects and Significance

Nowadays, with the increasing popularity of open-source software, the volume of open-source code is growing at an unprecedented[1] rate. Whether in enterprises or research institutions, more and more developers choose to copy and paste existing code to improve software development efficiency. The rapid development of major open-source communities has attracted countless users to share and collaborate on open-source projects, leading to the creation of numerous derivative[2] software applications that serve production, daily life, and scientific research. The introduction of

[1] unprecedented: 史无前例的

[2] derivative: 衍生

open-source code offers two main advantages.

On one hand, the direct use of existing code significantly enhances productivity and reduces costs. On the other hand, renowned open-source projects are typically developed and maintained by top-tier[3] developers and companies worldwide, ensuring high software quality and standardized structure.

[3] top-tier: 顶层的

However, as software continues to evolve and its functionalities expand, the negative impact of duplicated and cloned code on software quality, usability[4], and maintainability[5] becomes increasingly apparent. Code imported from open-source projects diminishes[6] developers' understanding and control over the overall software system. Conflicts may arise between the external code and the system's native code, and vulnerabilities[7] in the open-source code may be inadvertently[8] introduced into the project through copying, leading to the following issues.

[4] usability: 可用性
[5] maintainability: 可维护性
[6] diminish: 减少
[7] vulnerability: 脆弱性
[8] inadvertently: 无意地

(1) **Increased additional development costs**: In open-source projects, factors such as high specialization and incomplete documentation often require developers to spend more time understanding the code, thereby raising additional development costs. Moreover, the large volume of code in open-source projects means that reusing extensive associated code during cloning can inflate[9] the total code base after development, resulting in longer compilation[10] times and higher memory requirements.

[9] inflate: 膨胀
[10] compilation: 编译

(2) **Higher risk of vulnerabilities in developed software**: While open-source projects generally exhibits high quality and stability, they may still contain undiscovered potential vulnerabilities. These vulnerabilities can increase system risks and compromise[11] security.

[11] compromise: 妥协

(3) **Potential infringement[12] of open-source software copyrights**: When using open-source projects, compliance with open-source licenses (e.g., GPL, BSD, Apache License) is required. For instance, copyright issues must be considered when using code from GitHub. Unauthorized use of open-source code during cloning may violate[13] software copyrights and lead to legal[14] consequences.

[12] infringement: 侵权
[13] violate: 违反
[14] legal: 合法的

To address these issues, researchers often employ code similarity detection techniques to identify similar code in software engineering.

Code similarity detection is one of the fundamental tasks in the field of software engineering, playing a critical role in plagiarism[15] detection, license violation detection, software reuse analysis, and vulnerability discovery.

[15] plagiarism: 剽窃

2. Classification of Code Similarity Detection Approaches

The field of code similarity detection has witnessed significant academic progress since the 1970s, with numerous detection tools and

methodologies[16] being developed. As open-source code repositories[17] continue to expand exponentially[18], the need for large-scale similarity detection solutions has become increasingly imperative[19]. Modern large-scale detection systems serve multiple critical functions: they assist developers in managing open-source components[20] within projects, enable tracing of software component origins, facilitate[21] efficient code search operations, and allow analysis of component citation[22] frequency. These capabilities collectively contribute to enhanced software quality assurance and significant reductions in both development and maintenance expenditures[23]. Contemporary code similarity detection approaches are typically categorized into five distinct methodological tiers: text-based, lexical-based[24], syntax-based[25], semantic-based[26] and metric-based[27] techniques.

[16] methodology: 方法论
[17] repository: 仓库
[18] exponentially: 以指数方式
[19] imperative: 迫切的
[20] component: 组成部分
[21] facilitate: 促进
[22] citation: 引用
[23] expenditure: 花费
[24] lexical-based: 基于词法的
[25] syntax-based: 基于语法的
[26] semantic-based: 基于语义的
[27] metric-based: 基于度量值的

(1) Text-based Detection Methodology

Text-based detection represents the earliest approach to code similarity analysis. The methodology follows a systematic process: first, preprocess the code snippets[28], such as removing spaces, comments, etc; next, convert the code segments[29] into characters. If the characters of two code segments are the same, then the two code segments are the same. The advantage of this type of method is that the algorithm implementation process is simple and can be used to detect the source code of almost all programming languages; the disadvantage is that it cannot recognize the syntax, semantics, and other information of the program, resulting in low detection accuracy.

[28] snippet: 一小片段
[29] segment: 段

(2) Lexical-based Detection Method

Lexical-based detection method, also known as token based detection method. Firstly, parse[30] the code snippet into a sequence of strings; next, check the token sequences in different code snippets. If there are identical token sub-sequences, it indicates the existence of code cloning. Common detection algorithms include Longest Common Sub-sequence (LCS), suffix[31] tree matching, Karp Rabin fingerprinting algorithm, semantic indexing technology, etc. The advantage of this type of method is that it can use lightweight tools, which can be extended to detect code and plain text[32] in multiple programming languages, and has lower spatiotemporal complexity compared to complex detection algorithms based on syntax, semantics, etc; its disadvantage is similar to text-based detection methods, which cannot recognize logical information such as syntax and semantics of the program, resulting in lower detection accuracy.

[30] parse: 对……进行语法分析
[31] suffix: 后缀
[32] plain text: 纯文本

(3) Syntax-based Detection Method

Syntax-based detection method, also known as tree-based detection method. Firstly, by performing lexical and syntactic analysis on the code, an abstract syntax tree of the source program is constructed; next, compare the same or similar sub-trees to determine if code cloning has been performed. The advantage of this type of method is that it can recognize the syntax information of the program and improve detection accuracy compared to text-based or token based detection methods; the disadvantage is that the cost of constructing AST and matching language sub trees is high, and as the program size expands, the time and space complexity of the final detection method will be very high.

(4) Semantic-based Detection Method

Semantic-based detection method, is also known as graph-based detection method. Firstly, by analyzing the syntax structure and contextual[33] environment of the code, a program dependency graph of the source program is constructed; next, matching algorithms and program slicing are used to obtain identical or similar sub-graph isomorphic[34] PDGs (Program Dependent Graphs), and then it is determined whether cloning operations have been performed. The advantage of this type of method is that it can recognize the semantic logic information of the program and improve detection accuracy; its disadvantage is that the cost of constructing PDG and sub-graph isomorphic PDG is relatively high, and with the expansion of program size, the time and space complexity of detection methods are constantly increasing.

[33] contextual: 上下文的

[34] isomorphic: 同构的

(5) Metric-based Detection Method

The metric-based detection method is only applicable to fixed granularity[35] detection of code. Firstly, divide the code into fixed granularity comparison code units; next, extract metrics from the comparison code units to determine whether code cloning has been performed. Metrics include code variables, parameters, return values, etc. The advantage of metric-based detection methods is high detection accuracy and ease of code refactoring; its disadvantage is that it is limited to fixed granularity detection, and if the granularity is too large, the missed detection rate will be very high.

[35] granularity: 粒度

3. Code Similarity Detection Tool

There are detection methods based on text representation, such as Dup method, Duploc method, and NICAD method. Although these methods have the advantages of low cost and low computational

overhead, they are only suitable for simple clone detection of fully cloned types and cannot complete complex type detection. Lexical-based detection techniques include CCFinder (X), D-CCFinder, CP-Miner, and CCLearner. Although these methods improve the utilization of source code information, they still ignore the structural information of the source code during the detection process.

The methods based on syntax tree and metric are the main representation methods based on syntax detection technology. Famous tree based detection methods include CloneDR, Deckard, and CDLH, while indicator[36] based detection methods include the methods proposed by Mayrand et al. and Kontogiannis et al. These detection methods not only provide detection accuracy, but also bring inevitable problems. For example, tree-based detection methods have high overhead due to the need to traverse[37] the tree structure, while indicator based detection methods cannot guarantee high accuracy.

[36] indicator: 指示器

[37] traverse: 穿过

At present, the detection technologies based on semantic representation are graph-based detection technology and hybrid[38] technology. Famous graph based detection techniques include the methods proposed by Komon door et al. and the Duplix method, while well-known hybrid techniques include the ConQAT method. In recent years, semantic based deep learning methods have been widely studied and applied, with representative ones including Code2Vec, Tree-CNN, and Func2Vec. Based on current research results, semantic representation based detection techniques have shown good detection performance for all four types of clones, but there are also certain limitations, such as the limited applicability of AST representation methods to tasks and difficulty in transferring them to other tasks; the cost of constructing program dependency graphs (PDGs) and isomorphic sub-graphs is relatively high, and as the program size increases, the time and space complexity of detection methods also continue to rise.

[38] hybrid: 混合的

The Davey method is a metric-based detection method.

Task 3 Read the passage above and then translate the expressions into Chinese or English in pairs.

1. code similarity detection _____
2. plagiarism detection _____
3. suffix tree matching _____
4. vulnerability discovery _____
5. preprocess the code snippets _____
6. longest common subsequence _____
7. semantic indexing technology _____
8. spatiotemporal complexity _____
9. 软件著作权 _____
10. 基于语义 _____
11. 基于语法 _____
12. 基于度量值 _____
13. 检测准确率 _____
14. 基于词法 _____

15. 轻量级算法 16. 纯文本

1.3 Hot Topic Reading: Neuromorphic Computing

Deep Neural Networks (DNNs) emerged in the mid-1980s and have been driving the development of artificial intelligence since 2006. However, with the rapid advancement of edge intelligence in recent years, traditional Artificial Neural Networks (ANNs) have revealed several shortcomings. First, the success of ANNs has historically relied on massive training datasets and extensive GPU computational resources. For instance, the GPT-3 model boasts over 170 billion parameters and was trained on 45TB of data. Second, traditional ANNs lack biological interpret-ability[1] and dynamic[2] mechanisms within neurons, resulting in weak capabilities for processing spatiotemporal[3] information. Finally, ANN models are computationally inefficient and challenging to implement on hardware, particularly portable devices. These limitations have spurred the emergence of the third generation of neural networks —Spiking Neural Networks[4] (SNNs). Research shows that at the 45nm technology node, a multiply-accumulate operation in an ANN neuron consumes 4.6pJ, whereas a spiking neuron's accumulate operation consumes only 0.9pJ, making SNNs more suitable for deployment on low-power edge devices.

[1] biological interpret-ability: 生物可解释性
[2] dynamic: 动态的
[3] spatiotemporal: 时空的
[4] Spiking Neural Networks: 脉冲神经网络

Neuromorphic computing[5] is a biologically inspired computational paradigm that seeks to mimic the functions of neurons and synapses[6] in both temporal and spatial[7] domains based on spike-events in the brain, enabling highly energy-efficient computation. SNNs closely resemble[8] biological neural systems, using discrete values (spikes) to encode and process data, offering an efficient and low-power computational solution that aligns with[9] the core principles of neuromorphic computing. In biological neurons, a spike is generated when the cumulative[10] change in membrane[11] potential from presynaptic stimuli[12] exceeds a threshold[13]. The rate of spike generation and the temporal patterns of spike sequences carry information about external stimuli and ongoing computations. In SNNs, spiking neurons process information only when new input spikes arrive, making SNNs inherently more biologically plausible[14] and exhibiting advantageous features similar to real neural circuits, such as analog[15] computing capabilities, low power consumption, rapid inference[16], event-driven operation, online learning, and massive parallelism[17]. These advantages make SNNs highly suitable for deployment on edge devices. Currently, SNNs are the core model of

[5] neuromorphic computing: 神经形态计算
[6] synapse: 突触
[7] spatial: 空间的
[8] resemble: 像
[9] align with: 与……一致
[10] cumulative: 累积
[11] membrane: 膜电位
[12] presynaptic stimuli: 突触前刺激
[13] threshold: 阈值
[14] plausible: 合理的
[15] analog: 模拟
[16] inference: 推理
[17] parallelism: 排比

neuromorphic computing, holding the potential to fully leverage[18] various neuromorphic hardware platforms, enabling "near-memory computing" and breaking the bottlenecks of traditional Von Neumann[19] architectures.

In recent years, significant progress has been made in neuromorphic computing, but the edge deployment of SNNs still faces several unresolved challenges. This paper will analyze and summarize the limitations and challenges in current research and propose potential solutions.

1. Deficiencies in Spiking Neural Network Models

Despite their superior energy efficiency and biological plausibility, SNN models still have inherent limitations.

(1) Trade-off[20] between Computational Performance and Biological Fidelity[21] in Spiking Neurons

Although the widely adopted Leaky Integrate-and-Fire (LIF) model offers computational efficiency, its biological plausibility is lower compared to other spiking neuron models. Thus, balancing the learning capabilities and biological realism of spiking neurons remains a critical research topic.

To address this challenge, exploring new learning rules inspired by biological principles or refined mathematical modeling could lead to better-designed spiking neurons. Alternatively, task-specific spiking neurons can be developed. For example, Tempo-tron neurons, due to their temporal sensitivity, excel in processing sequential data[22] and are well-suited for speech or signal recognition tasks.Evolvable spiking neurons, which adaptively adjust spike rates, are more suitable for power-constrained edge devices.

(2) Limitations of Rate Coding and Temporal Coding

Rate coding and temporal coding each have strengths and weaknesses. Rate coding focuses only on spike counts within a time window, ignoring inter-spike interference[23] and failing to fully utilize spatiotemporal information in spike sequences. In contrast, many temporal coding schemes better exploit timing information but rely on complex synaptic functions, increasing power consumption and hindering edge deployment.

A potential solution is combining rate and temporal coding, adopting appropriate encoding methods across network layers or between neurons based on task requirements. Current research has begun exploring this direction, with hopes for deeper advancements. Additionally, designing

[18] leverage: 杠杆作用

[19] Von Neumann: 冯·诺依曼

[20] trade-off: 矛盾

[21] fidelity: 可信度

[22] sequential data: 时序数据

[23] inter-spike interference: 尖峰间干扰

hardware-compatible spike encoders is another promising avenue.

(3) Challenges in Training Spiking Neural Networks

Training SNNs effectively has always been a focal point in SNN research. While the sparsity[24] of spikes grants SNNs significant power advantages, the non-differentiation[25] of spike events prevents direct application of back-propagation[26], and no universally optimal training method exists yet. For edge deployment scenarios[27]where high-compute training platforms are unavailable—especially if online learning is required—metrics[28] like memory resource utilization and weight update complexity must be considered alongside[29] traditional performance indicators[30]. Thus, selecting suitable training algorithms is a core challenge for SNN edge deployment[31].

[24] sparsity: 稀疏
[25] non-differentiation: 不可微性
[26] back-propagation: 反向传播
[27] scenario: 场景
[28] metric: 指标
[29] alongside: 在……旁边
[30] indicator: 指标
[31] deployment: 部署

Addressing this requires long-term optimization of existing training methods. Variants of Spike-Timing-Dependent Plasticity (STDP) and spike-based back-propagation are evolving, and ANN-to-SNN conversion methods now achieve accuracy comparable to traditional ANNs. Furthermore, integrating advanced biological mechanisms into SNN training is a promising direction for balancing biological plausibility[32] and learning capabilities.

[32] plausibility: 合理性

(4) Inherent Flaws in Static Model Architectures

Many current SNN models combine traditional ANNs (e.g., CNNs and RNNs) with SNNs. While these hybrid[33] models inherit[34] the strengths of traditional architectures in processing image and sequential data while retaining low-power spike-based communication, they fail to fully address issues like high training complexity, weak interpretability[35],and poor tolerance in hardware fault in such complex ANN models.

[33] hybrid: 混合的
[34] inherit: 继承
[35] interpretability: 可解释性

Exploring dynamic-topology SNNs is a viable solution[36]. Models like Evolving Spiking Neural Networks (ESNNs) adaptively adjust network structures, leveraging the unique advantages of neuromorphic hardware platforms and better suiting embedded edge scenarios.

[36] viable solution: 可行解决方案

2. Challenges in Neuromorphic Hardware Platform Design

Beyond SNN model limitations, neuromorphic hardware platforms face significant design hurdles.

(1) High Barrier to Hardware Programming

For example, programming on FPGA platforms requires AI researchers to understand hardware description languages and circuit principles, and the lack of mature software frameworks like TensorFlow

or PyTorch raises development barriers. For neuromorphic chips and novel non-volatile[37] memory devices, developers must use specialized tools and sometimes even possess knowledge of physical materials.

[37] non-volatile：非易失性的

Developing comprehensive neuromorphic computing tool-chains is crucial for SNN edge deployment. Efficient tool-chains can streamline development and facilitate[38] research. Existing SNN software frameworks and simulation tools need continuous updates for better compatibility[39] and stability[40]. High-level Synthesis[41] (HLS) tools, which convert high-level code into hardware descriptions[42] and automatically generate circuits, can lower FPGA deployment barriers. Additionally, designing scalable[43] hardware mapping methods is a promising approach.

[38] facilitate: 促进
[39] compatibility: 兼容性
[40] stability: 稳定性
[41] synthesis: 合成
[42] description: 描述
[43] scalable: 可扩展的

(2) Performance Bottlenecks in Neuromorphic Hardware

Most neuromorphic hardware platforms are low-power, small-area devices with limited resources like memory, making them unsuitable for large-scale, complex models. Moreover, traditional CMOS circuits are constrained by 2D connections and limited interconnect metals/routing protocols, posing challenges in emulating 3D biological brain structures—a major bottleneck for current neuromorphic chips.

Future directions include exploring advanced fabrication[44] processes to transcend CMOS limitations, developing more efficient memristor arrays[45] , or designing hybrid architectures that combine ANNs and SNNs to improve overall performance.

[44] fabrication: 制造
[45] memristor array: 忆阻器阵列

(3) Hardware Platform Reliability Issues

Studies show that neural networks, including SNNs—often assumed to inherit the fault tolerance of biological brains—exhibit limited robustness in recent fault injection experiments. Since underlying hardware support is not fully reliable, many SNN deployments fail to realize the full potential of neuromorphic hardware.

To enhance edge deployment reliability, solutions may involve designing more stable neuromorphic hardware inspired by biology or improving SNN models with advanced fault-tolerance mechanisms to bolster[46] robustness[47] in edge AI applications.

[46] bolster: 支持
[47] robustness: 鲁棒性

1.4 Listening: Data Representation

Unit 1

Task 4 Listen to the tape with some blanks for you to fill in.

At the end of last lecture, we started introducing some of the pieces you want to do. And I want to remind you of our goal. We're trying to describe processes. We want

to have things that deduce new kinds of information. So we want to write programs to do that. If we're going to write programs, we need at least two things: we need some 1._____ for fundamental data. And we saw last time two examples of that. And the second thing we're going to need, is we're going to need a way to give 2._____ to the computer to 3._____ that data. We need to give it a description of the recipe. In terms of primitive data, what we saw were two kinds. Right? 4._____ and 5._____. A little later on in the lecture we're going to introduce a third kind of value, but what we're going to see throughout the term is, no matter how complex a data structure we create, and we're going to create a variety of data structures, 6._____ all of them have their basis, their atomic level if you like, are going to be some combinations of numbers, of strings, and the third type, which are 7._____, which I'm going to introduce a little later on in this lecture. And that kind of makes sense right? Numbers are there to do 8._____ things, strings are our fundamental way of representing 9._____ information. And so we're going to see how to combine those things as we go along. Second thing we saw was, we saw that associated with every 10._____ value was a 11._____. And these are kind of obvious, right? Strings are strings. For numbers,we had some 12._____; we had 13._____, we had 14._____.We'll introduce a few more as we go along. But those types are important, because they tell us something about what we want to do when we want to put them together. OK, but nonetheless, I want to stress we have both a 15._____, yeah, and a type. All right. Once we have them, we want to start making 16._____ out of them. We want to put pieces together. And for that, we combine things in expressions. And what we saw as expressions are formed of 17._____ and 18._____. And the simple things we did were the sort of things you'd expect from numerical things. Now I want to stress one other nuance (细微差别) here, which we some examples of this. Initially we just typed in expressions into the 19._____; that is, directly into Python. And as I suggested last time, the interpreter is actually a program inside of the machine that is basically following the rules we're describing here to deduce the value and print it up. And if we type directly from into the 20._____, it essentially does an eval and a print. It 21._____, and it prints. Most of the time, we're going to be doing expressions inside of some piece of code, inside of a 22._____, which is the Python word for program. In there, I want to make this distinction, this nuance: the evaluator is still going to be taking those expressions and using its rules to get a value, but it's not going to print them back out. Why? Because typically, you're doing that to use it somewhere else in the program. It's going to be stored away in a 23._____. It's going to be stuck in a data structure. It's going to be used for a side effect. So, inside of code, or inside of a script, there's no 24._____, unless we make it 25._____. And that's a little bit down in the weeds.

1.5 Writing: How to Write a Personal Resume and Practise Based on the Given Examples

The outline for a resume structured the order of the resume headings. Each heading is followed by a paragraph and the information you provide within it. To build your resume, you have to decide

on its format and its sections as the frame of your resume outline. This article provides sample resume outlines—examples of resume structure and options for headings. It also discusses the information to include within each section and sub-sections. The outline of the resume forms the type of the resume you select, which depends on your professional situation.

Mastering a Resume Outlines

Contact Information:

- Name
- Address: Street, City, State, Zip
- Phone Number
- Email Address

Summary of Qualifications: Highlight your key skills, accomplishments, and experiences relevant to the desired position. Focus on what makes you a strong candidate and briefly mention your career objectives.

Core Competencies: List your core competencies or areas of expertise. Include specific skills, knowledge, and abilities that align with the job requirements.

Professional Experience:

Job Title: Company Name, Location (Dates)

- Describe your responsibilities, achievements, and contributions in this role. Use bullet points to clearly outline your accomplishments and quantify them whenever possible.
- Provide a brief summary of your responsibilities and accomplishments for each position you held. Emphasize transferrable skills and experiences.

Education:

Degree: Major, Institution Name, Location (Year)

Include relevant degrees, certifications, and diplomas. Mention any honors, awards, or special achievements.

Additional Training and Certifications:

List any professional development courses, workshops, or certifications that enhance your qualifications for the targeted position. Include the institution, location, and year of completion.

Skills:

- Technical Skills: List specific technical skills relevant to the job, such as programming languages, software proficiency, or industry-specific tools.
- Transferable Skills: Highlight soft skills that demonstrate your ability to adapt, communicate, problem-solve, and work effectively in a team or leadership role.

Languages:

Indicate your language proficiency, including fluency in multiple languages if applicable.

References:

- Available upon request.
- State that references can be provided upon request. Ensure you have obtained permission from the individuals you intend to use as references.

A Sample

The following professionals can use this resume example as a draft: Computer Specialist/ Programmer Analyst, Computer Systems Analyst/Programmer and Computer Systems Programmer Analyst.

Job Description:Computer programmers design a set of programs that acts as instructions for a computer to process information. The work of a programmer depends upon the type of project he/she is assigned and the complexity of the work. According to the objective or the job description, he prepares the flow charts.

Computer Applications Programmer Resume Example

[Full Name]

[Street, City, State, Zip]

[Phone Number]

[Email Address]

COMPUTER APPLICATIONS PROGRAMMER

Accomplished computer programmer with 9 years' experience in .NET software applications and an in-depth knowledge of programming languages for development and programming; technical expertise includes:

J2EE, MS.NET, SQL SERVER | Oracle/SQL | ASP Controls/ .NET 3.5
SAP | C# .NET 3.5 MICROSOFT VISUAL STUDIO |
ASP.NET 3.5 | JAVA

Functional Skills

- Critical reasoning to forecast the strength and weakness of a program.
- Effectively handle concurrent multiple assignments.
- Detail-oriented and highly articulate.
- Excellent problem solving skills.

Professional Experience

BYT S/W Applications Wilmington, DE 2006—Present

Applications Programmer

Worked on several challenging projects related to education, government and medical fields. Consulted with clients and engineers to solve problems and recommend changes in programs.

- Assessed the objective of the project by studying the job description, and prepared flow charts accordingly.
- Coordinated with system analysts and software engineers in completing complex projects.
- Monitored the program and made required changes to produce smooth and efficient run.
- Conducted tests and trial runs to ensure that the program produced the desired results in a

timely manner.

- Documented the program and clarified it with instructions, resulting in easily comprehended coding.

Selected Achievements

- Successfully completed (X) projects in the past year.
- Managed to increase the running speed of (X) programs by (X)%.
- Changed (X) method to (Y) method in order to increase efficiency and customer satisfaction.
- Initiated and completed a project that resulted in increased revenues of (X).

Educational Qualifications

Massachusetts Institute of Technology Cambridge, MA	2001—2005
Computer Science & Computer Engineering, Technology	
Saint Cloud Technical College, BSc.	1998—2000

Task 5 Write a resume according to your own experience and check up your writings with your partner.