Unions and Structs

```
module = Module(
    code="ELEE1147",
    name="Programming for Engineers",
    credits=15,
    module_leader="Seb Blair BEng(H) PGCAP MIET MIHEEM FHEA"
)
```


Download as a PDF

What are Structs and Unions?

- **Structs**: Composite data types that group variables under a single name.
- **Unions**: Similar to structs but share the same memory space for all members.

Struct

- A struct is a user-defined data type in which different data types can be grouped together under a single name.

- It is declared using the `struct` **keyword**, followed by the structure's name and a block of members enclosed in curly braces `{...}`.

```
struct [NameOfStruct] {
    char var1;
    int var2;
    float var3;
};
```

```
struct Person {
    char name[50];
    int age;
    float height;
};
```

Accessing Members

- **Accessing Members with Pointers**
  - The arrow operator (`->`) is used when accessing members through pointers.
  - or `(*structName).member`

```c
struct Person {
    char name[50];
    int age;
    float height;
};

int main(){

  struct Person person1;
  struct Person *personPointer = &person1;

  // Using the arrow operator
  personPointer->age = 25;

    // Equivalent longhand notation
  (*personPointer).age = 25;

  return 0;
}
```

Arrays of Structs and Struct Members

- **Array of Structs**
  - You can create an array of structs to manage multiple records efficiently.

```c
struct Student {
  char name[50];
  int age;
  float grades[5]; // Member is an array
};

int main(){

  int n = 20; // Number of students
  struct Student *class = (struct Student *)malloc(n * sizeof(struct Student));

  class[0].age = 35;
  strcpy(class[0].name, "Marshall Bruce Mathers III");
  class[0].grades[0] = 90.5;

  return 0;
}
```

Arrays of Structs and Struct Members

- **Array of Structs**
  - Size of the struct is:

$$
\begin{aligned}
50\,\text{Bytes} &= 1 \times 50\,\text{Bytes}\,\texttt{char} \\
4\,\text{Bytes} &= 1 \times 4\,\text{Bytes}\,\texttt{int} \\
20\,\text{Bytes} &= 4 \times 5\,\text{Bytes}\,\texttt{float} \\
&\downarrow \\
74\,\text{Bytes} &=
\end{aligned}
$$

```c
struct Student {
  char name[50];
  int age;
  float grades[5]; // Member is an array
};

int main(){

  int n = 20; // Number of students
  struct Student *class = (struct Student *)malloc(n * sizeof(struct Student));

  class[0].age = 35;
  strcpy(class[0].name, "Marshall Bruce Mathers III");
  class[0].grades[0] = 90.5;

  return 0;
}
```

Arrays of Structs and Struct Members

- **Array of Structs**
  - Size of the struct array is:

$$\begin{aligned}\text{Total Bytes} &= n \times \text{sizeof(struct Student)} \\ &= 20 \times \text{sizeof(struct Student)} \\ &= 20 \times 74\,\text{Bytes} \\ &= 1480\,\text{Bytes}\end{aligned}$$

```c
struct Student {
  char name[50];
  int age;
  float grades[5]; // Member is an array
};

int main(){

  int n = 20; // Number of students
  struct Student *class = (struct Student *)malloc(n * sizeof(struct Student));

  class[0].age = 35;
  strcpy(class[0].name, "Marshall Bruce Mathers III");
  class[0].grades[0] = 90.5;

  return 0;
}
```

Unions

Definition:

- Members share the same memory location.
- Size of union is the size of the largest member.

```
union Data {
    int intValue;
    float floatValue;
    char stringValue[20];
};
```

Properties of Unions

- **Memory Sharing**
  - All members of a union share the same memory space.
  - Size of the union is determined by the largest member.

- **Single-Access at a Time**
  - Only one member of the union can be accessed at any given time.
  - Modifying one member affects the value of the others.

Type Conversion with Unions

- **Memory Overlay:**
  - Unions store all their members at the same memory location.
  - This means that if you write a value to one member, you are essentially modifying the same memory that is used by other members.
- **Type Interpretation:**
  - When you access a member of the union, you interpret the stored bits according to the type of that member.
  - For example, if you write an integer to a union member and then access another member of type float, you interpret the same bits as a floating-point number.

```
❯ ./unions2.exe
sizeof(union Data): 4 bytes
Address of union Data: 00000025FA5FFCBC

After setting intValue = 42:
  intValue: 42
  floatValue: 0.000000
  stringValue: *
Memory content (hex):
  Byte  0: 0x2A
  Byte  1: 0x00
  Byte  2: 0x00
  Byte  3: 0x00

After setting floatValue = 3.14:
  intValue: 1078523331
  floatValue: 3.140000
  stringValue: ├┘H@`Γ┘
Memory content (hex):
  Byte  0: 0xC3
  Byte  1: 0xF5
  Byte  2: 0x48
  Byte  3: 0x40

After setting stringValue = "One":
  intValue: 6647375
  floatValue: 0.000000
  stringValue: One
Memory content (hex):
  Byte  0: 0x4F
  Byte  1: 0x6E
  Byte  2: 0x65
  Byte  3: 0x00
```

⚠️ **Caution:**

- It's crucial to be cautious when using unions for type conversion to avoid undefined behavior or unexpected results.

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

union Data {
    int intValue; float floatValue; char stringValue[4];
};

void printUnionBytes(const union Data *d) {
    const unsigned char *p = (const unsigned char *)d;
    printf("Memory content (hex):\n");
    for (size_t i = 0; i < sizeof(union Data); ++i) {
        printf("  Byte %2zu: 0x%02X\n", i, p[i]);
    }
    printf("\n");
}

void printUnionState(const union Data *d) {
    printf("  intValue: %d\n", d->intValue);
    printf("  floatValue: %f\n", d->floatValue);
    printf("  stringValue: %s\n", d->stringValue);
}

int main() {
    union Data d;

    printf("sizeof(union Data): %zu bytes\n", sizeof(d));
    printf("Address of union Data: %p\n\n", (void*)&d);

    d.intValue = 42;
    printf("After setting intValue = 42:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    d.floatValue = 3.14f;
    printf("After setting floatValue = 3.14:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    strcpy(d.stringValue, "Hello, world!");
    printf("After setting stringValue = \"Hello, world!\":\n");
    printUnionState(&d);
    printUnionBytes(&d);

    return 0;
}
```

```
❯ ./unions2.exe
sizeof(union Data): 4 bytes
Address of union Data: 00000025FA5FFCBC

After setting intValue = 42:
  intValue: 42
  floatValue: 0.000000
  stringValue: *
Memory content (hex):
  Byte  0: 0x2A
  Byte  1: 0x00
  Byte  2: 0x00
  Byte  3: 0x00

After setting floatValue = 3.14:
  intValue: 1078523331
  floatValue: 3.140000
  stringValue: ├┘H@`Γ┘
Memory content (hex):
  Byte  0: 0xC3
  Byte  1: 0xF5
  Byte  2: 0x48
  Byte  3: 0x40

After setting stringValue = "One":
  intValue: 6647375
  floatValue: 0.000000
  stringValue: One
Memory content (hex):
  Byte  0: 0x4F
  Byte  1: 0x6E
  Byte  2: 0x65
  Byte  3: 0x00
```

⚠ **Caution:**

- It's crucial to be cautious when using unions for type conversion to avoid undefined behavior or unexpected results.

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

union Data {
    int intValue; float floatValue; char stringValue[4];
};

void printUnionBytes(const union Data *d) {
    const unsigned char *p = (const unsigned char *)d;
    printf("Memory content (hex):\n");
    for (size_t i = 0; i < sizeof(union Data); ++i) {
        printf("  Byte %2zu: 0x%02X\n", i, p[i]);
    }
    printf("\n");
}

void printUnionState(const union Data *d) {
    printf("  intValue: %d\n", d->intValue);
    printf("  floatValue: %f\n", d->floatValue);
    printf("  stringValue: %s\n", d->stringValue);
}

int main() {
    union Data d;

    printf("sizeof(union Data): %zu bytes\n", sizeof(d));
    printf("Address of union Data: %p\n\n", (void*)&d);

    d.intValue = 42;
    printf("After setting intValue = 42:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    d.floatValue = 3.14f;
    printf("After setting floatValue = 3.14:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    strcpy(d.stringValue, "Hello, world!");
    printf("After setting stringValue = \"Hello, world!\":\n");
    printUnionState(&d);
    printUnionBytes(&d);

    return 0;
}
```

```
❯ ./unions2.exe
sizeof(union Data): 4 bytes
Address of union Data: 00000025FA5FFCBC

After setting intValue = 42:
  intValue: 42
  floatValue: 0.000000
  stringValue: *
Memory content (hex):
  Byte  0: 0x2A
  Byte  1: 0x00
  Byte  2: 0x00
  Byte  3: 0x00

After setting floatValue = 3.14:
  intValue: 1078523331
  floatValue: 3.140000
  stringValue: ├┘H@`Γ┘
Memory content (hex):
  Byte  0: 0xC3
  Byte  1: 0xF5
  Byte  2: 0x48
  Byte  3: 0x40

After setting stringValue = "One":
  intValue: 6647375
  floatValue: 0.000000
  stringValue: One
Memory content (hex):
  Byte  0: 0x4F
  Byte  1: 0x6E
  Byte  2: 0x65
  Byte  3: 0x00
```

⚠️ **Caution:**

- It's crucial to be cautious when using unions for type conversion to avoid undefined behavior or unexpected results.

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

union Data {
    int intValue; float floatValue; char stringValue[4];
};

void printUnionBytes(const union Data *d) {
    const unsigned char *p = (const unsigned char *)d;
    printf("Memory content (hex):\n");
    for (size_t i = 0; i < sizeof(union Data); ++i) {
        printf("  Byte %2zu: 0x%02X\n", i, p[i]);
    }
    printf("\n");
}

void printUnionState(const union Data *d) {
    printf("  intValue: %d\n", d->intValue);
    printf("  floatValue: %f\n", d->floatValue);
    printf("  stringValue: %s\n", d->stringValue);
}

int main() {
    union Data d;

    printf("sizeof(union Data): %zu bytes\n", sizeof(d));
    printf("Address of union Data: %p\n\n", (void*)&d);

    d.intValue = 42;
    printf("After setting intValue = 42:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    d.floatValue = 3.14f;
    printf("After setting floatValue = 3.14:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    strcpy(d.stringValue, "Hello, world!");
    printf("After setting stringValue = \"Hello, world!\":\n");
    printUnionState(&d);
    printUnionBytes(&d);

    return 0;
}
```

```
❯ ./unions2.exe
sizeof(union Data): 4 bytes
Address of union Data: 00000025FA5FFCBC

After setting intValue = 42:
  intValue: 42
  floatValue: 0.000000
  stringValue: *
Memory content (hex):
  Byte  0: 0x2A
  Byte  1: 0x00
  Byte  2: 0x00
  Byte  3: 0x00

After setting floatValue = 3.14:
  intValue: 1078523331
  floatValue: 3.140000
  stringValue: ├┘H@`Γ┘
Memory content (hex):
  Byte  0: 0xC3
  Byte  1: 0xF5
  Byte  2: 0x48
  Byte  3: 0x40

After setting stringValue = "One":
  intValue: 6647375
  floatValue: 0.000000
  stringValue: One
Memory content (hex):
  Byte  0: 0x4F
  Byte  1: 0x6E
  Byte  2: 0x65
  Byte  3: 0x00
```

⚠ Caution:

- It's crucial to be cautious when using unions for type conversion to avoid undefined behavior or unexpected results.

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

union Data {
    int intValue; float floatValue; char stringValue[4];
};

void printUnionBytes(const union Data *d) {
    const unsigned char *p = (const unsigned char *)d;
    printf("Memory content (hex):\n");
    for (size_t i = 0; i < sizeof(union Data); ++i) {
        printf("  Byte %2zu: 0x%02X\n", i, p[i]);
    }
    printf("\n");
}

void printUnionState(const union Data *d) {
    printf("  intValue: %d\n", d->intValue);
    printf("  floatValue: %f\n", d->floatValue);
    printf("  stringValue: %s\n", d->stringValue);
}

int main() {
    union Data d;

    printf("sizeof(union Data): %zu bytes\n", sizeof(d));
    printf("Address of union Data: %p\n\n", (void*)&d);

    d.intValue = 42;
    printf("After setting intValue = 42:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    d.floatValue = 3.14f;
    printf("After setting floatValue = 3.14:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    strcpy(d.stringValue, "Hello, world!");
    printf("After setting stringValue = \"Hello, world!\":\n");
    printUnionState(&d);
    printUnionBytes(&d);

    return 0;
}
```

```
❯ ./unions2.exe
sizeof(union Data): 4 bytes
Address of union Data: 00000025FA5FFCBC

After setting intValue = 42:
  intValue: 42
  floatValue: 0.000000
  stringValue: *
Memory content (hex):
  Byte  0: 0x2A
  Byte  1: 0x00
  Byte  2: 0x00
  Byte  3: 0x00

After setting floatValue = 3.14:
  intValue: 1078523331
  floatValue: 3.140000
  stringValue: ├┘H@`Γ┘
Memory content (hex):
  Byte  0: 0xC3
  Byte  1: 0xF5
  Byte  2: 0x48
  Byte  3: 0x40

After setting stringValue = "One":
  intValue: 6647375
  floatValue: 0.000000
  stringValue: One
Memory content (hex):
  Byte  0: 0x4F
  Byte  1: 0x6E
  Byte  2: 0x65
  Byte  3: 0x00
```

⚠️ **Caution:**

- It's crucial to be cautious when using unions for type conversion to avoid undefined behavior or unexpected results.

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

union Data {
    int intValue; float floatValue; char stringValue[4];
};

void printUnionBytes(const union Data *d) {
    const unsigned char *p = (const unsigned char *)d;
    printf("Memory content (hex):\n");
    for (size_t i = 0; i < sizeof(union Data); ++i) {
        printf("  Byte %2zu: 0x%02X\n", i, p[i]);
    }
    printf("\n");
}

void printUnionState(const union Data *d) {
    printf("  intValue: %d\n", d->intValue);
    printf("  floatValue: %f\n", d->floatValue);
    printf("  stringValue: %s\n", d->stringValue);
}

int main() {
    union Data d;

    printf("sizeof(union Data): %zu bytes\n", sizeof(d));
    printf("Address of union Data: %p\n\n", (void*)&d);

    d.intValue = 42;
    printf("After setting intValue = 42:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    d.floatValue = 3.14f;
    printf("After setting floatValue = 3.14:\n");
    printUnionState(&d);
    printUnionBytes(&d);

    strcpy(d.stringValue, "Hello, world!");
    printf("After setting stringValue = \"Hello, world!\":\n");
    printUnionState(&d);
    printUnionBytes(&d);

    return 0;
}
```

Where might you see a union?

Reason	Description	Example
Save memory	Only one value active at a time	Embedded devices
Reinterpret bits	View data as different types	Networking, hardware drivers
Model alternatives	Type-safe "choice" between types	Tagged union design