Variable Declaration and Assignment Statement

Programming in C

Shirley B. Chu
shirley.chu@delasalle.ph

College of Computer Studies
De La Salle University

November 8, 2021

Variables

- used to store values

Variables

- used to store values
- identified by a unique and descriptive *variable name*

Variables

- used to store values
- identified by a unique and descriptive *variable name*
- has a *type*

Variables

- used to store values
- identified by a unique and descriptive *variable name*
- has a *type*
- name of the memory location where the value is stored

Variable naming convention

- starts with a lowercase letter
  e.g. `nVal`, `fAmount`

Variable naming convention

- starts with a lowercase letter
  e.g. `nVal`, `fAmount`
- camel-cased
  e.g. `nNoOfStudents`, `fUnitPrice`

Variable naming convention

- starts with a lowercase letter
  e.g. `nVal`, `fAmount`
- camel-cased
  e.g. `nNoOfStudents`, `fUnitPrice`
- Hungarian notation : an identifier naming convention, where the type of data is indicated in the prefix of the variable name.

Type	Prefix	Example
`char`	`c`	`cChoice`
`int`	`n`	`nValue`

Type	Prefix	Example
`float`	`f`	`fPrice`
`double`	`d`	`dTotal`

Variable declaration

- Variables, in C, must *declared* before first use.

Variable declaration

- Variables, in C, must *declared* before first use.
- When writing a declaration statement, the *variable name* and the *type of data* must be indicated.

Variable declaration

- Variables, in C, must *declared* before first use.
- When writing a declaration statement, the *variable name* and the *type of data* must be indicated.

  e.g.
```
int nValue;
float fUnitPrice;
char cChoice, cAns;
```

Which data type to use?

- depends on the purpose of the variable

Which data type to use?

- depends on the purpose of the variable

  `int` for whole numbers, like class size, count

Which data type to use?

- depends on the purpose of the variable

  `int` for whole numbers, like class size, count
  `float` for real numbers, like amount, weight

Which data type to use?

- depends on the purpose of the variable

  `int` for whole numbers, like class size, count
  `float` for real numbers, like amount, weight

- depends on the range of the data type

Which data type to use?

- depends on the purpose of the variable

  `int` for whole numbers, like class size, count
  `float` for real numbers, like amount, weight

- depends on the range of the data type

Type	Size (bits)
`char`	8

Type	Size (bits)
`int`	32
`long`	64

Type	Size(bits)
`float`	32
`double`	64

Which data type to use?

- depends on the purpose of the variable

  `int` for whole numbers, like class size, count
  `float` for real numbers, like amount, weight

- depends on the range of the data type

Type	Size (bits)
`char`	8

Type	Size (bits)
`int`	32
`long`	64

Type	Size(bits)
`float`	32
`double`	64

  - To compute for the ranges,

	formula, where N is the number of bits	
	minimum value	maximum value
signed type	-2^{N-1}	$2^{N-1}-1$
unsigned type	0	$2^{N}-1$

Local variables

- *Local variables* are variables declared within a *function* or within a *block*.

Local variables

- *Local variables* are variables declared within a *function* or within a *block*.
- *Block* is a sequence of statements grouped together inside a pair of *braces*: { and }.

Local variables

- *Local variables* are variables declared within a *function* or within a *block*.
- *Block* is a sequence of statements grouped together inside a pair of *braces*: { and }.
- The *scope* of a local variable (*lifetime*) starts at its declaration, until the end of the block where it is declared.

Assignment Statement

=

- the *assignment operator*

Assignment Statement

=

- the *assignment operator*
- assigns a value to a variable

Assignment Statement

=

- the *assignment operator*
- assigns a value to a variable

Assignment Statement Syntax:

$$variable = value\_or\_expression;$$

Assignment Statement

=

- the *assignment operator*
- assigns a value to a variable

Assignment Statement Syntax:

variable = *value_or_expression*;

e.g.

```
nValue = 5;
fAmount = 3.5 * 9 / 3;
cValue = 'a' + 5;
```

Assignment Operator

highest	`()`	left to right
	unary `+`, unary `-` `!`	right to left
	`%`, `*`, `/`	left to right
	`+`, `-`	left to right
	`>`, `>=`, `<`, `<=`	left to right
	`==`, `!=`	left to right
	`&&`	left to right
	`||`	left to right
	`=`	right to left

C Operator Precedence Table: https://en.cppreference.com/w/c/language/operator_precedence

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

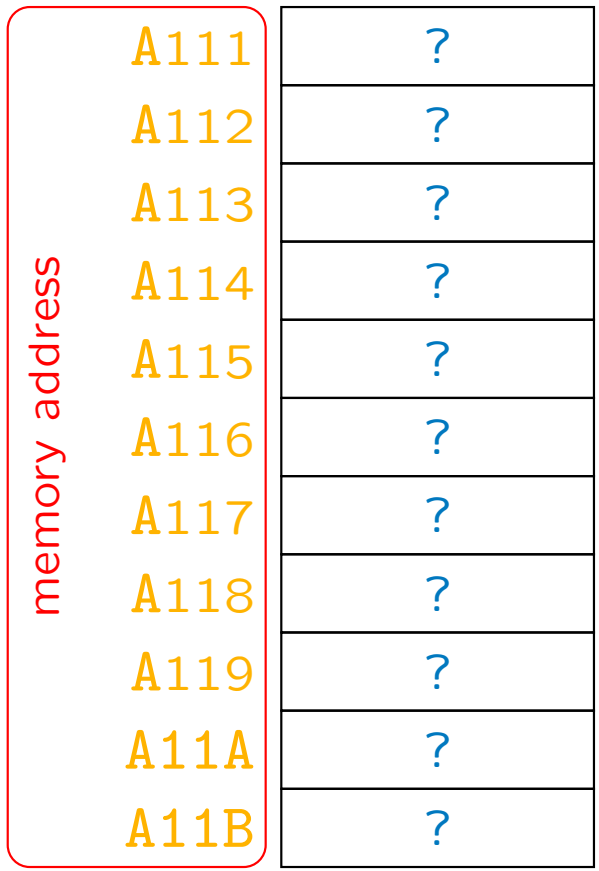
Memory

A111	?
A112	?
A113	?
A114	?
A115	?
A116	?
A117	?
A118	?
A119	?
A11A	?
A11B	?

Example 1

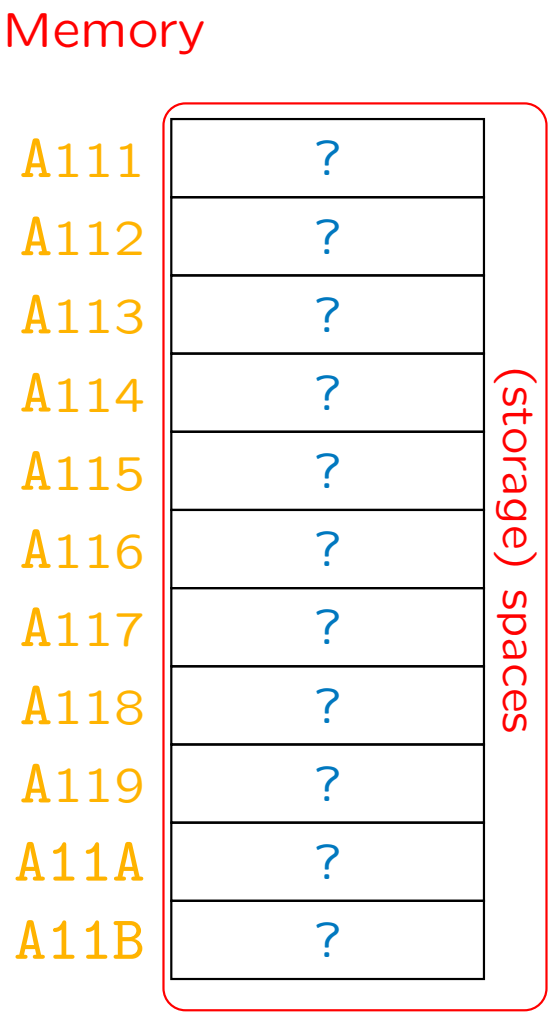
```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

Memory



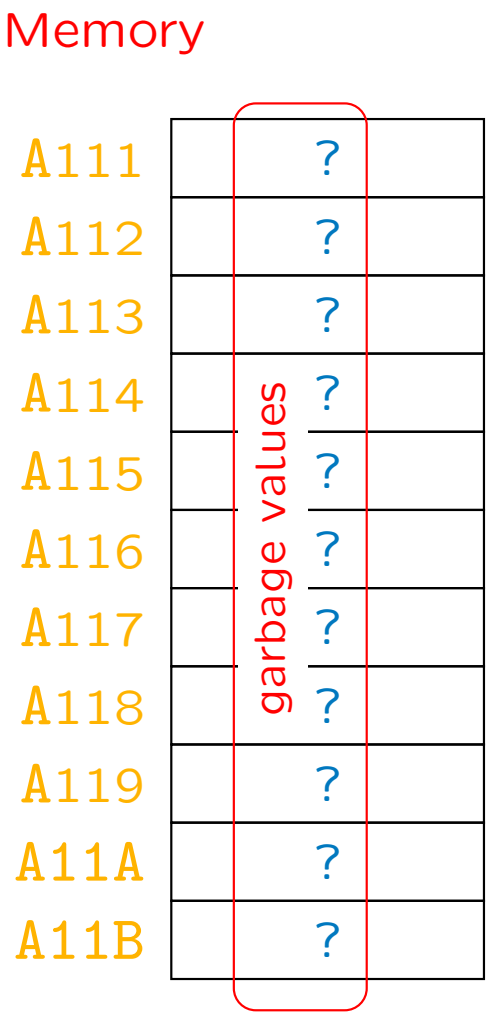
Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

Memory

A111	?
A112	?
A113	?
A114	?
A115	?
A116	?
A117	?
A118	?
A119	?
A11A	?
A11B	?

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

Memory

A111	?
A112	?
A113	?
A114	?
A115	?
A116	?
A117	?
A118	?
A119	?
A11A	?
A11B	?

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

Memory

A111	?
A112	?
A113	?
A114	?
A115	?
A116	?
A117	?
A118	?
A119	?
A11A	?
A11B	?

Example 1

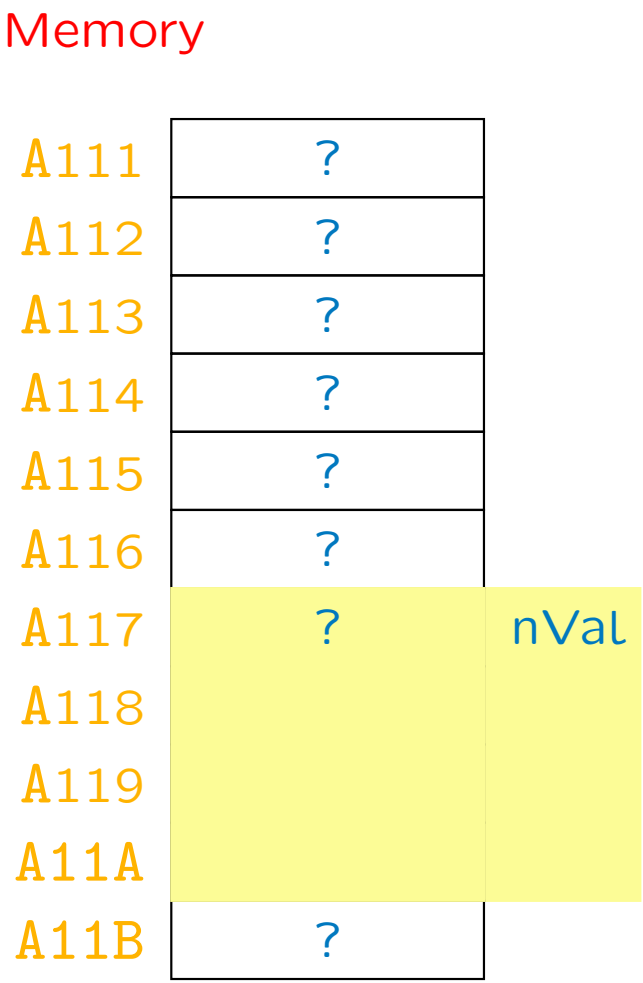
```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

Memory

A111	?
A112	?
A113	?
A114	?
A115	?
A116	?
A117	?
A118	?
A119	?
A11A	?
A11B	?

Example 1

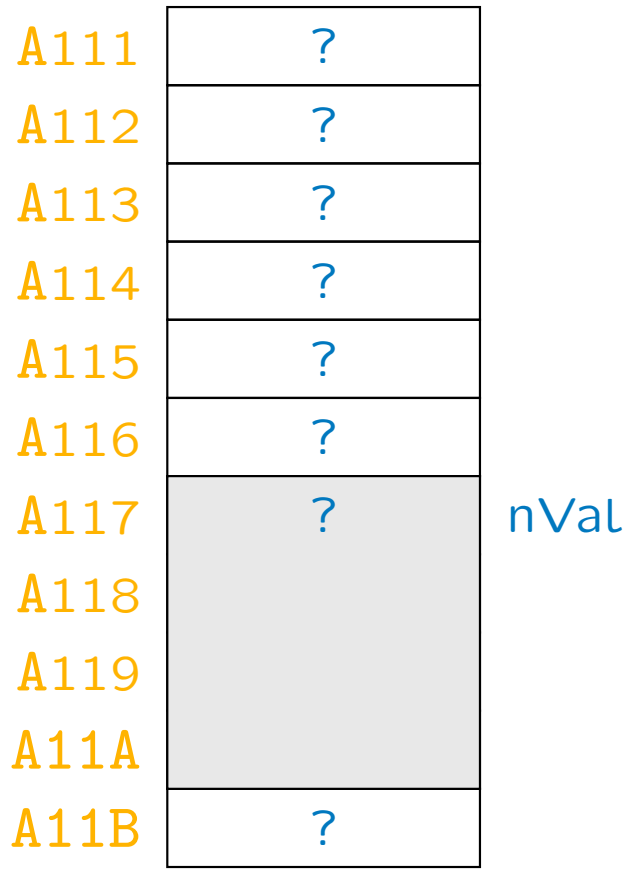
```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

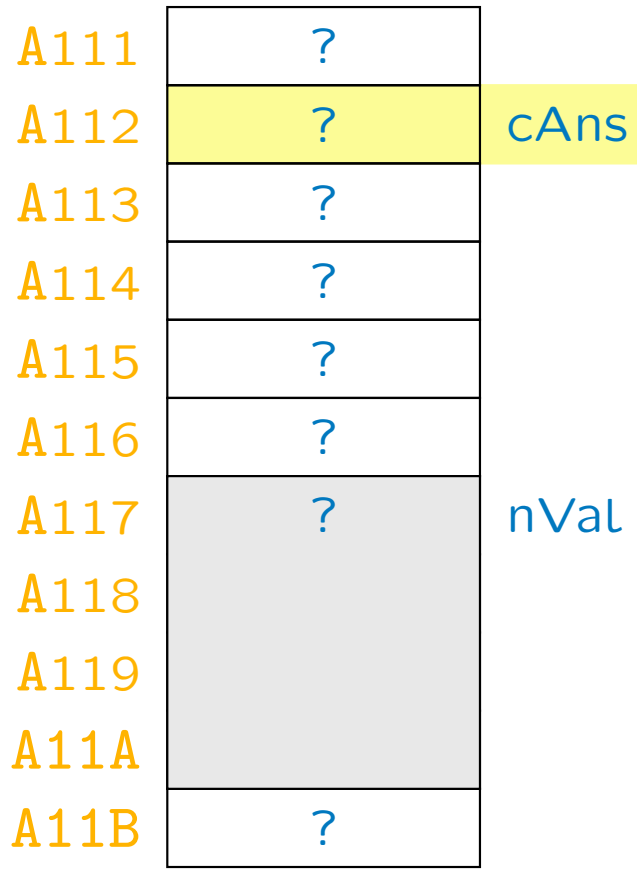
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

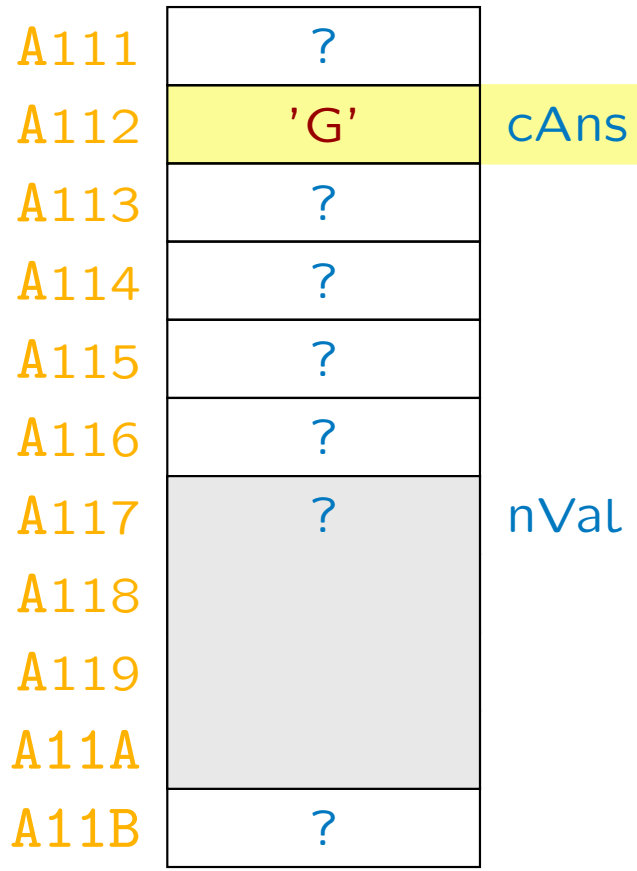
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

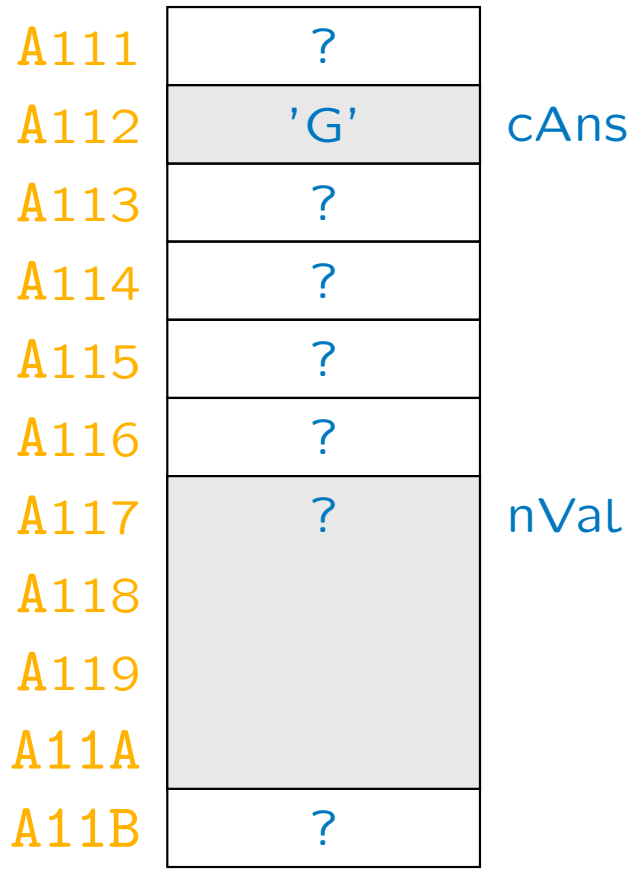
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

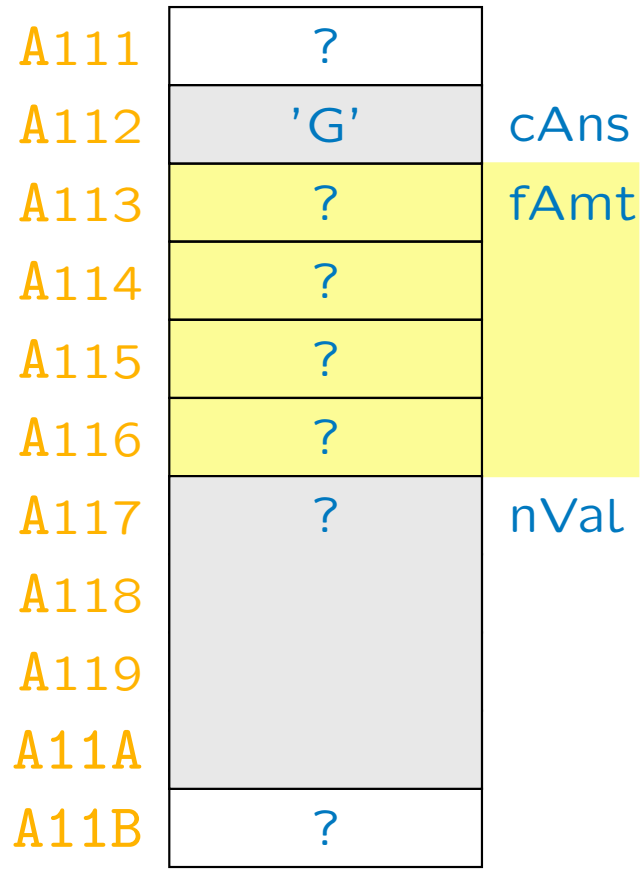
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

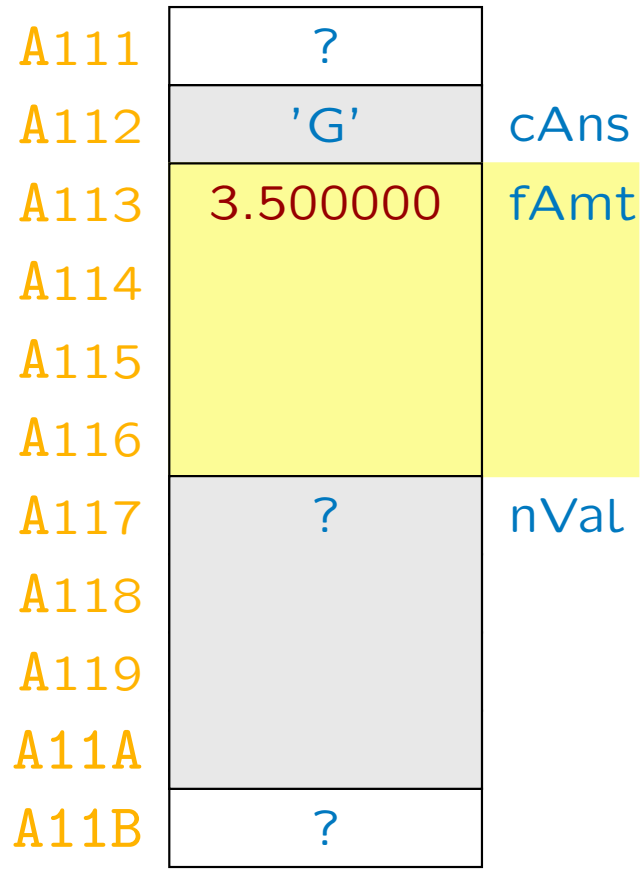
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

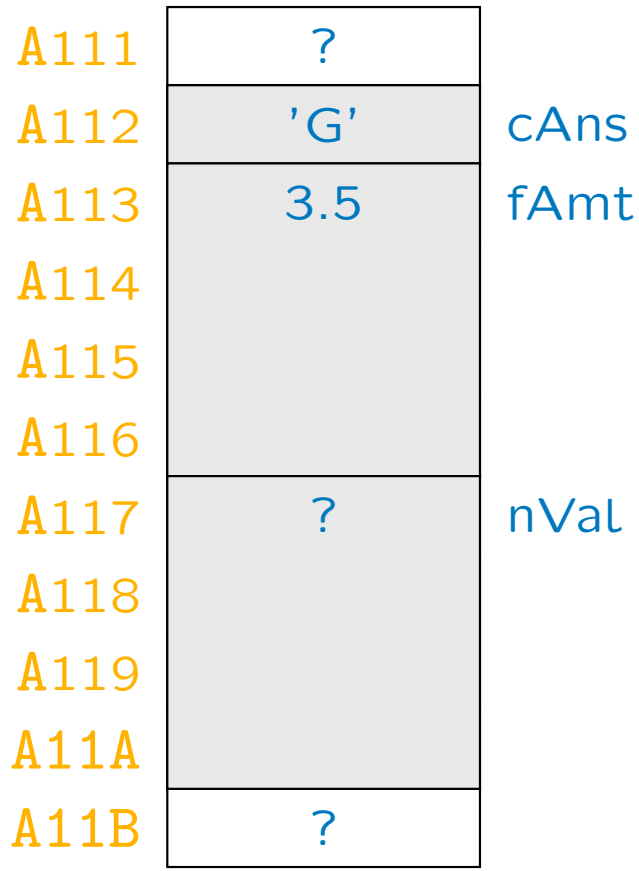
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

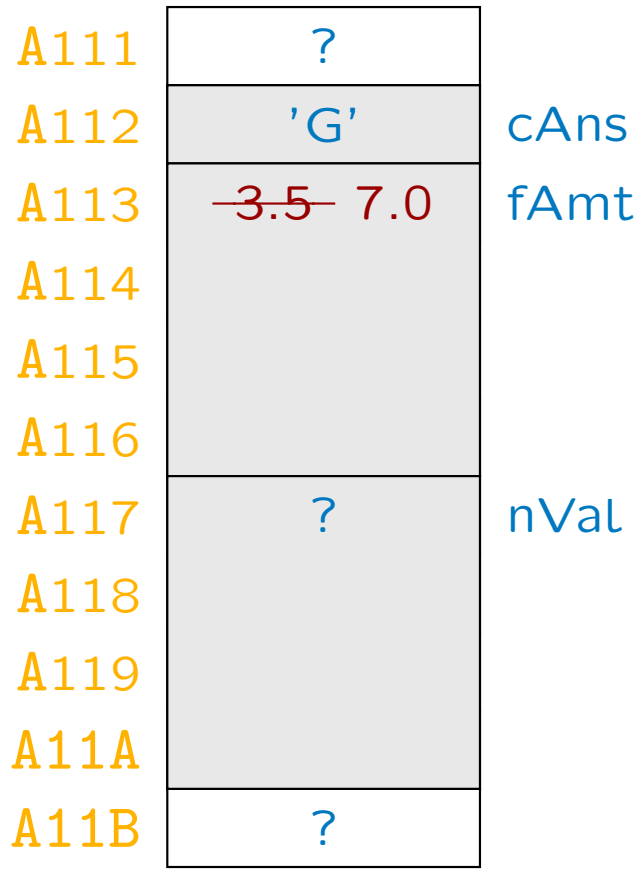
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

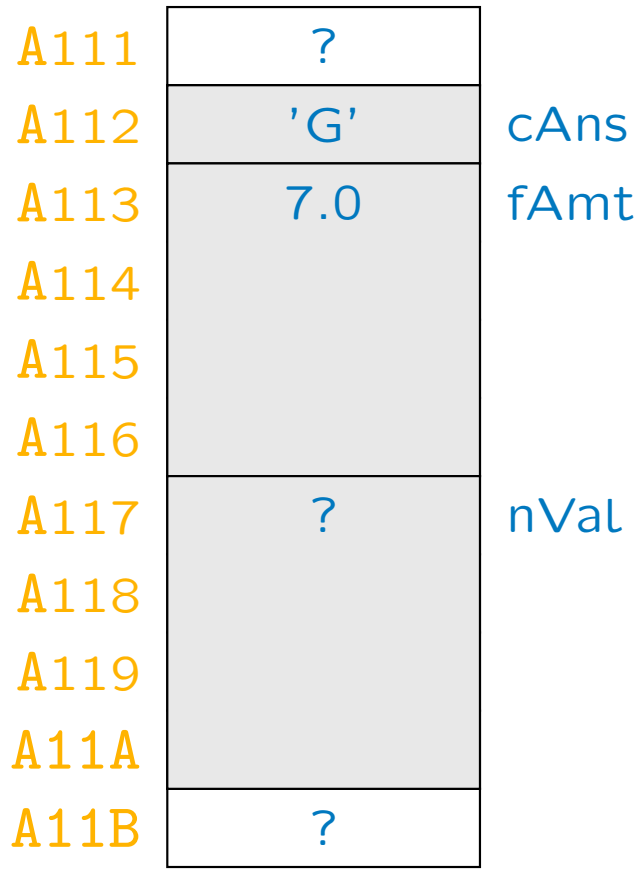
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

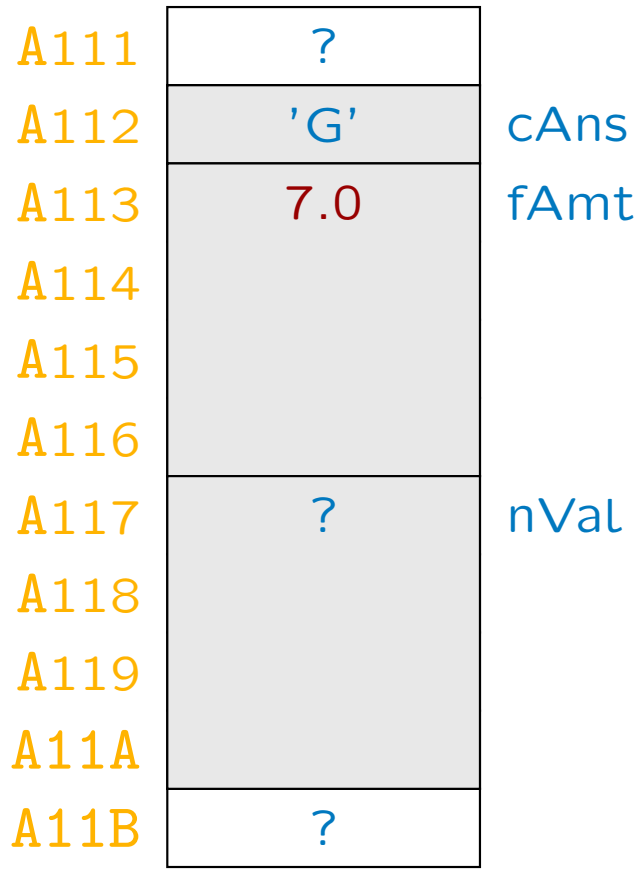
Memory



Example 1

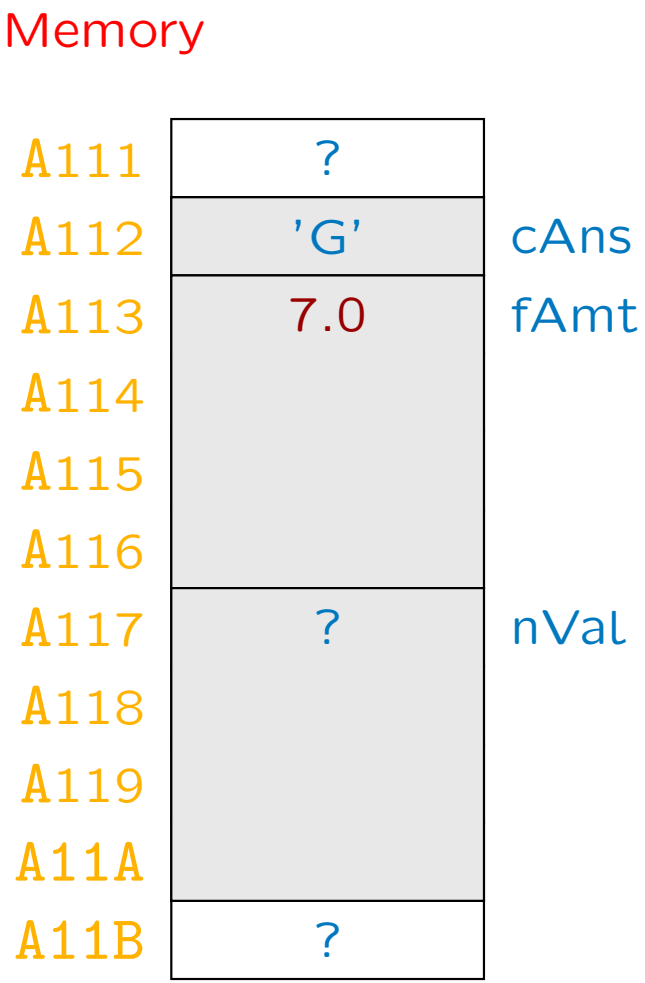
```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

Memory



Example 1

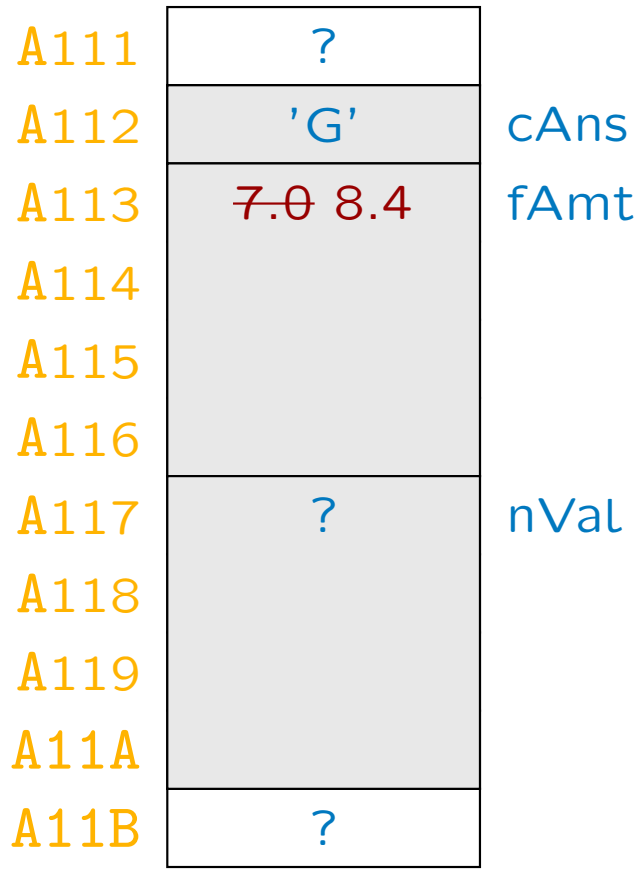
```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 7.0 * 1.2

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      //  8.4

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

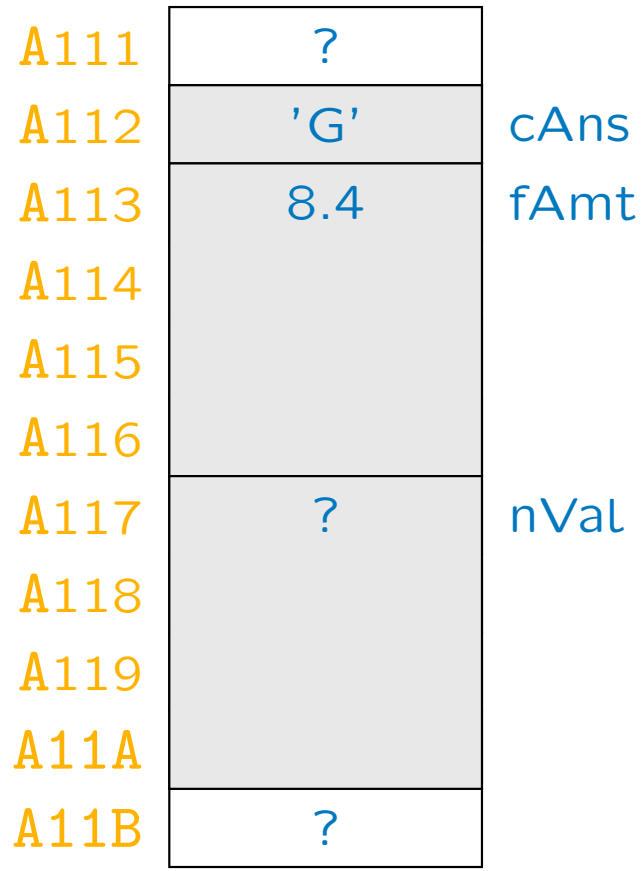
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;

    nVal = nVal - 3;

    return 0;
}
```

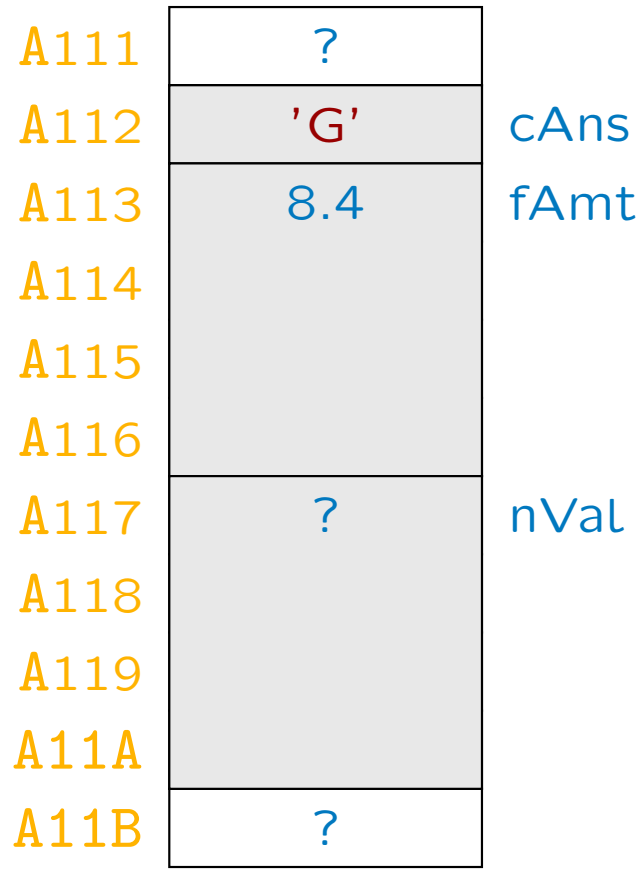
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'G' - 3

    nVal = nVal - 3;

    return 0;
}
```

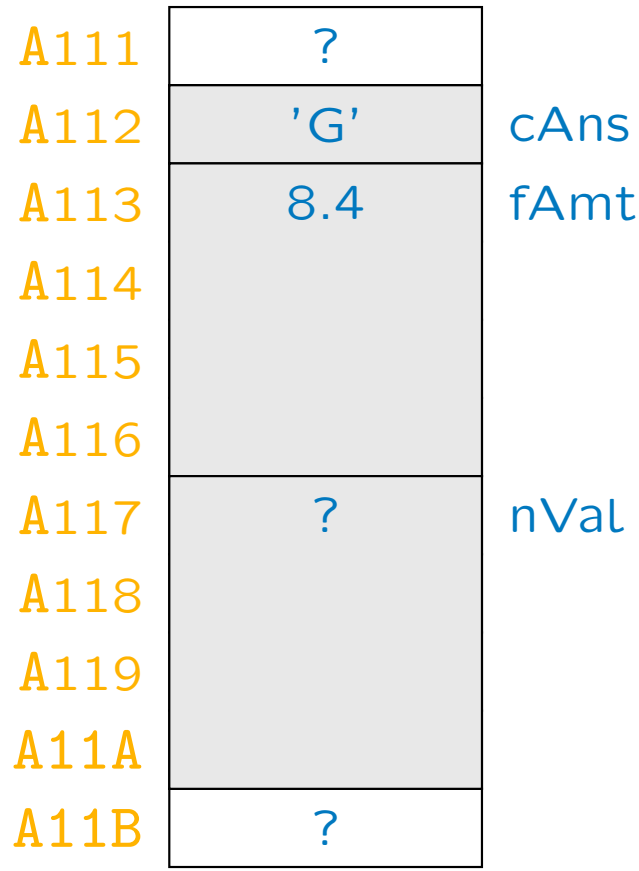
Memory



Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'G' - 3

    nVal = nVal - 3;

    return 0;
}
```

Memory

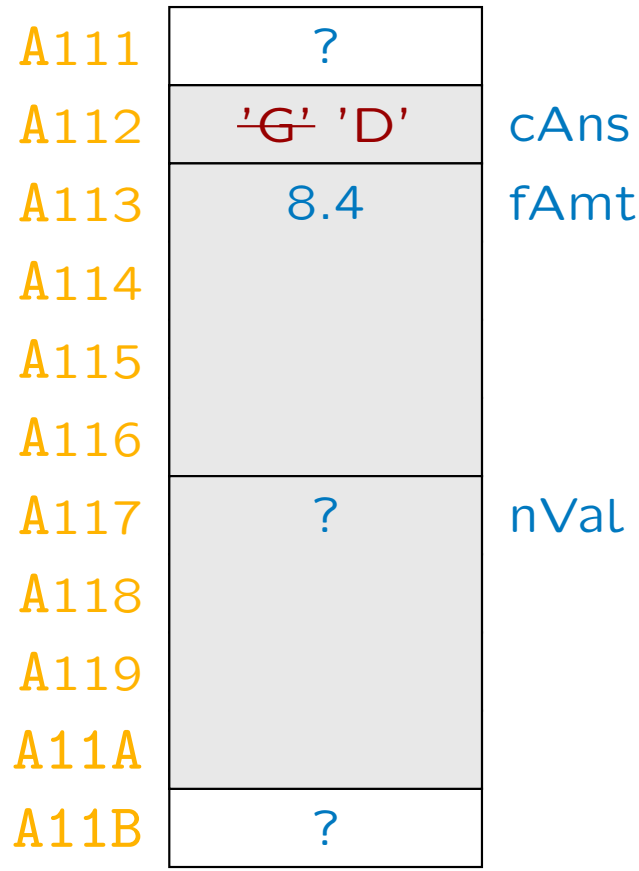


- ASCII value of 'A' is 65.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'

    nVal = nVal - 3;

    return 0;
}
```

Memory

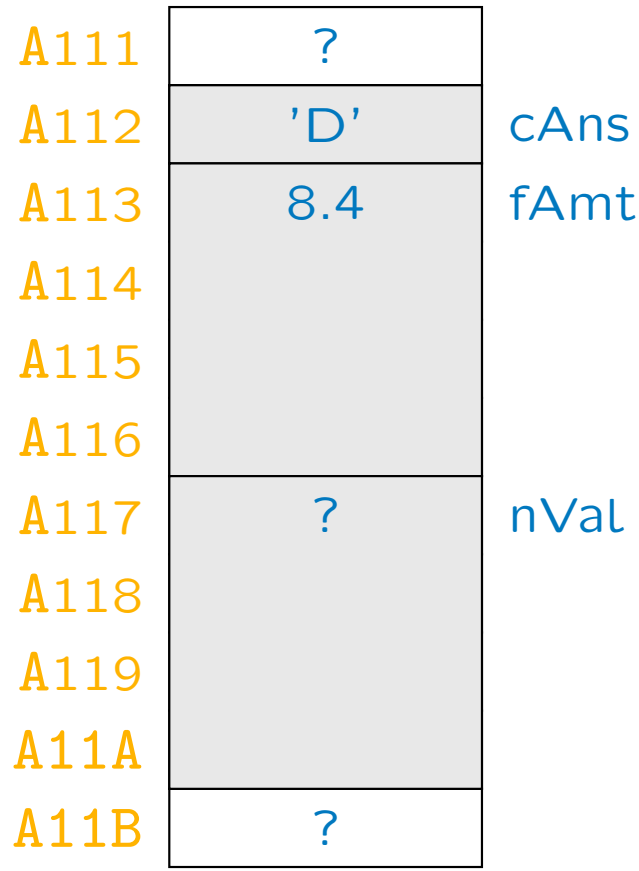


- ASCII value of 'A' is 65.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'

    nVal = nVal - 3;

    return 0;
}
```

Memory

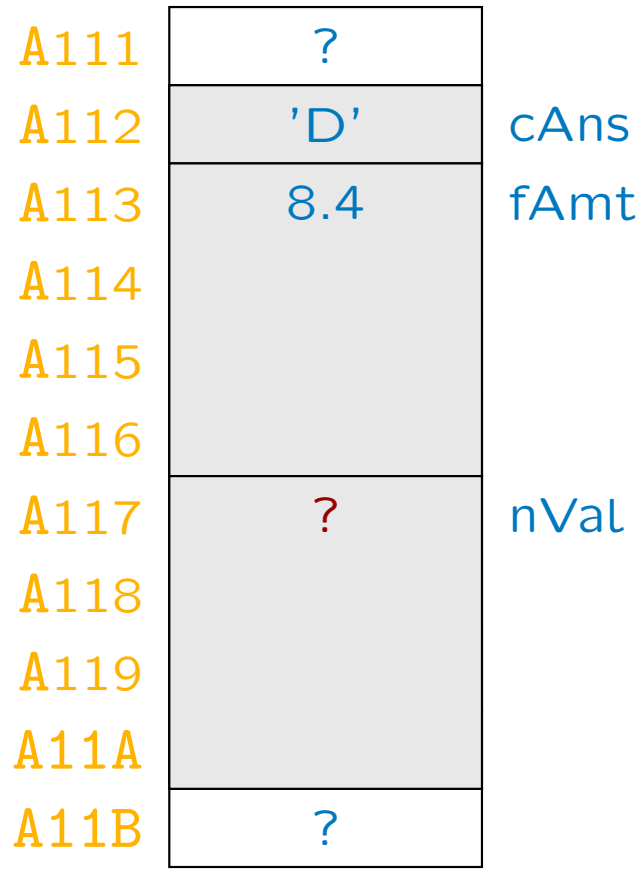


- ASCII value of 'A' is 65.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'

    nVal = nVal - 3;

    return 0;
}
```

Memory

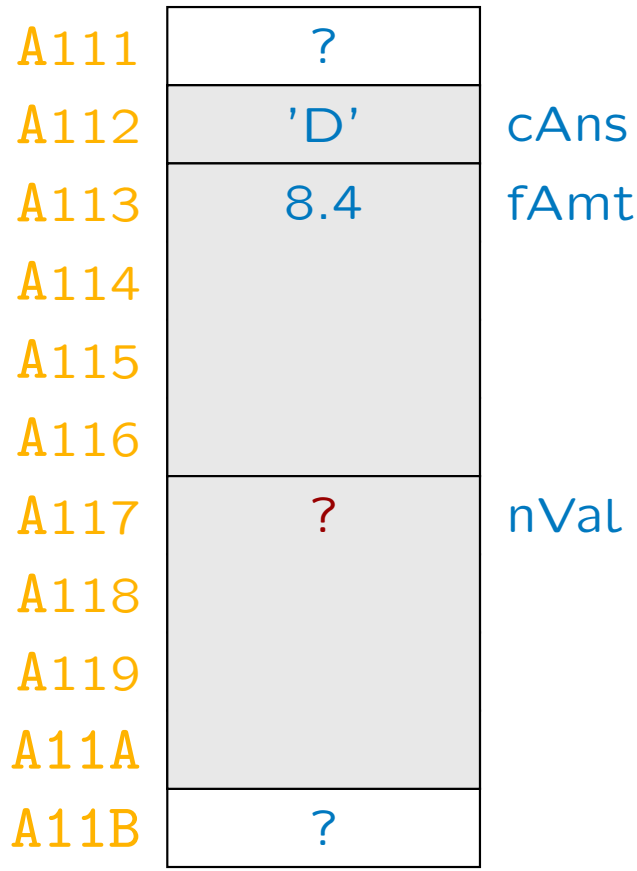


- ASCII value of 'A' is 65.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'

    nVal = nVal - 3;

    return 0;
}
```

Memory

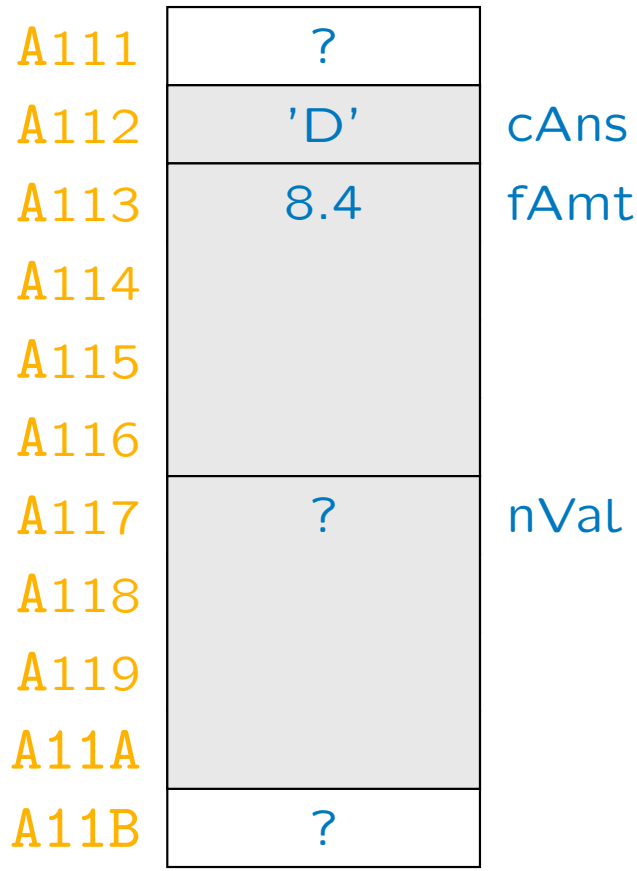


- ASCII value of 'A' is 65.
- Variables not initialized have *garbage* values.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'
    nVal = 5;
    nVal = nVal - 3;

    return 0;
}
```

Memory

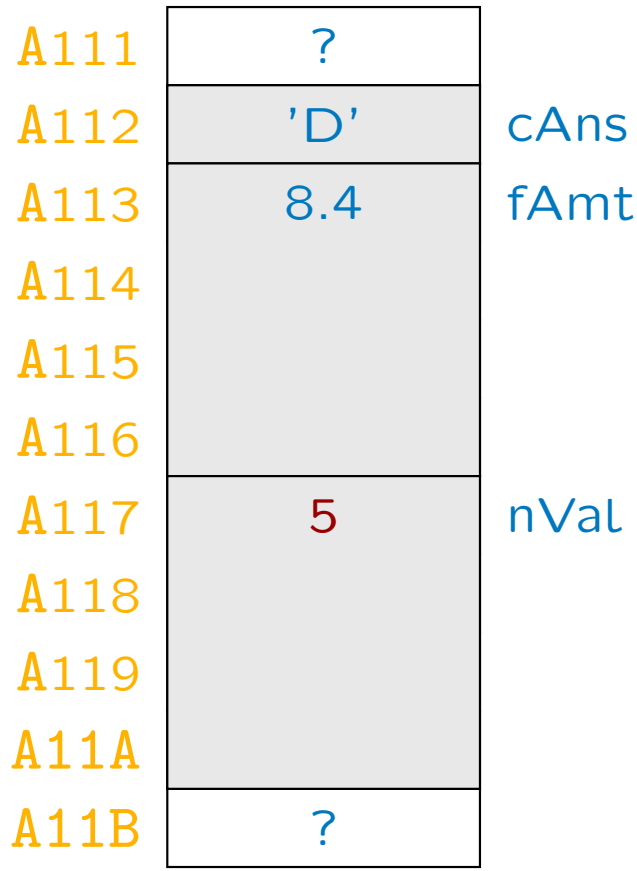


- ASCII value of 'A' is 65.
- Variables not initialized have *garbage* values.
- Variables that will be read first, MUST be initialized.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'
    nVal = 5;
    nVal = nVal - 3;

    return 0;
}
```

Memory

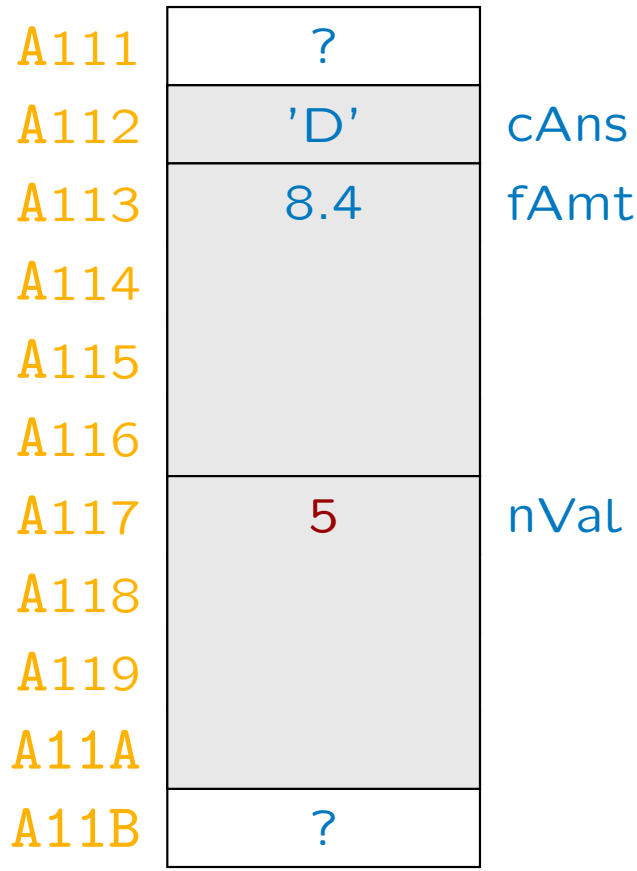


- ASCII value of 'A' is 65.
- Variables not initialized have *garbage* values.
- Variables that will be read first, MUST be initialized.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'
    nVal = 5;
    nVal = nVal - 3;

    return 0;
}
```

Memory

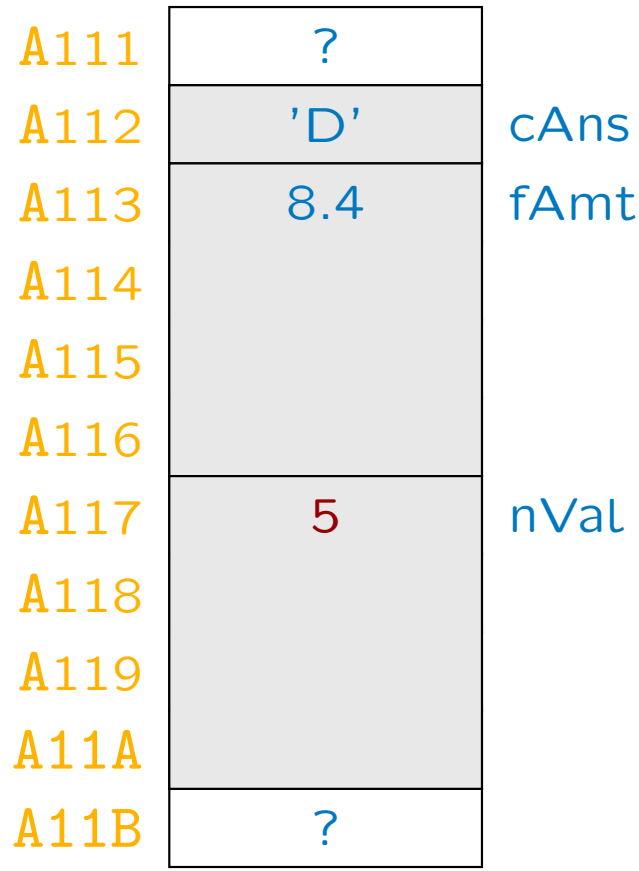


- ASCII value of 'A' is 65.
- Variables not initialized have *garbage* values.
- Variables that will be read first, MUST be initialized.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'
    nVal = 5;
    nVal = nVal - 3;        // 5 - 3

    return 0;
}
```

Memory

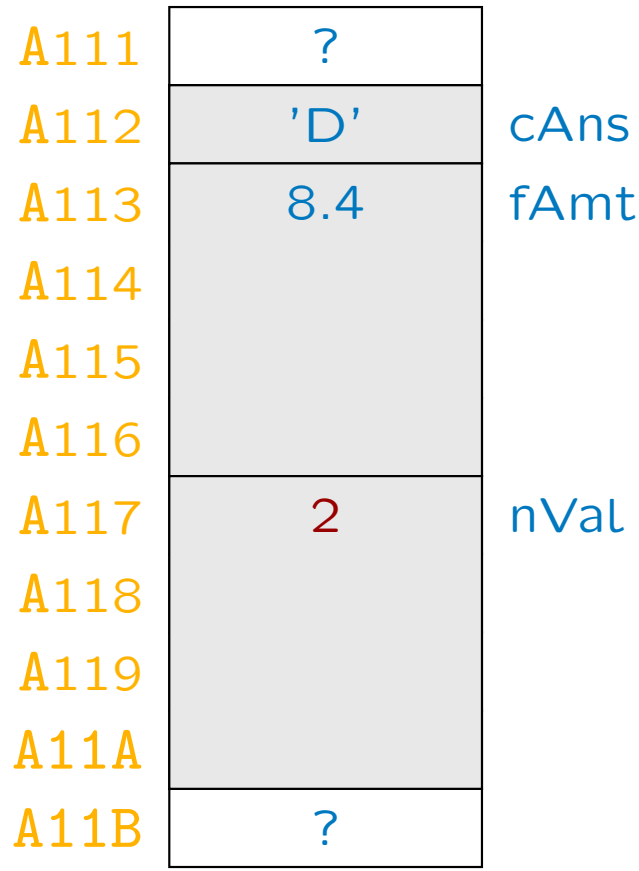


- ASCII value of 'A' is 65.
- Variables not initialized have *garbage* values.
- Variables that will be read first, MUST be initialized.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'
    nVal = 5;
    nVal = nVal - 3;        // 2

    return 0;
}
```

Memory

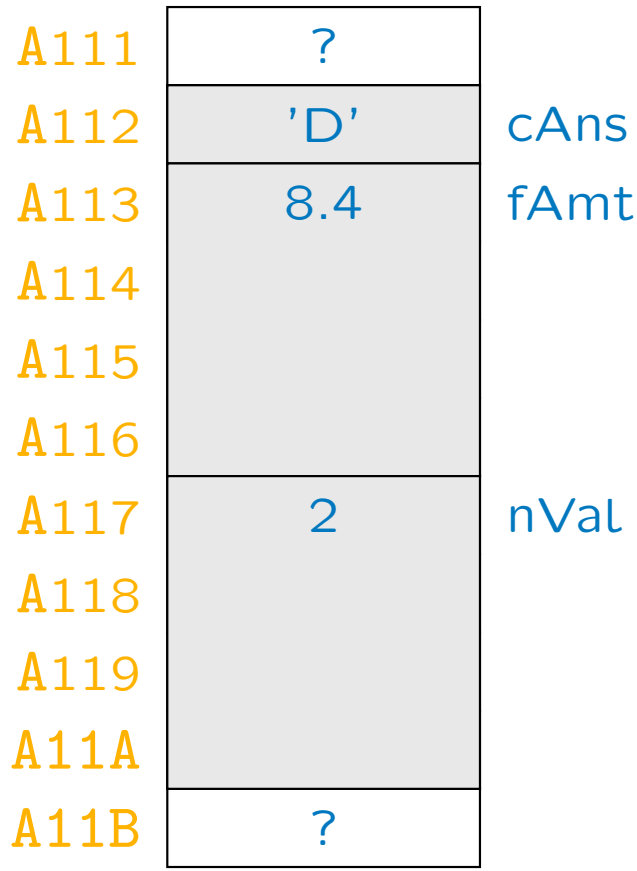


- ASCII value of 'A' is 65.
- Variables not initialized have *garbage* values.
- Variables that will be read first, MUST be initialized.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'
    nVal = 5;
    nVal = nVal - 3;        // 2

    return 0;
}
```

Memory

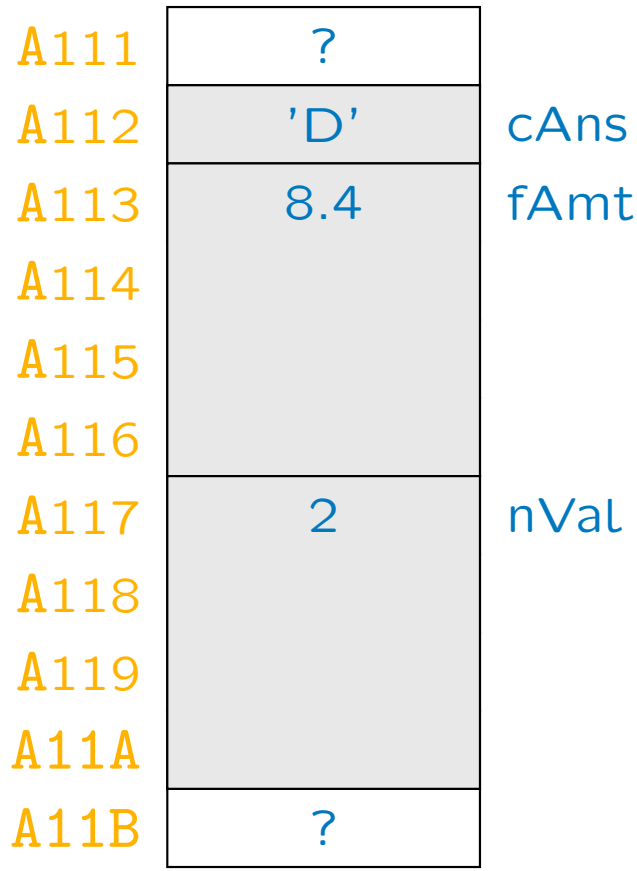


- ASCII value of 'A' is 65.
- Variables not initialized have *garbage* values.
- Variables that will be read first, MUST be initialized.

Example 1

```
int main()
{
    int nVal;
    char cAns = 'G';
    float fAmt = 3.5;

    fAmt = 7;
    fAmt = fAmt * 1.2;      // 8.4

    cAns = cAns - 3;        // 'D'
    nVal = 5;
    nVal = nVal - 3;        // 2

    return 0;
}
```

Memory



- ASCII value of 'A' is 65.
- Variables not initialized have *garbage* values.
- Variables that will be read first, MUST be initialized.

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y =$

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

$z = ?$

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

$z = ?$

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

$z = ?$

Note that = has right to left associativity.

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

$z = ?$

Note that = has right to left associativity.

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

$z = \cancel{?}\ 5$

Note that = has right to left associativity.

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

x = 5

y = 5

z = ~~?~~ 5

m =

Note that = has right to left associativity.

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

$z = \cancel{?}\ 5$

$m = 5$

Note that = has right to left associativity.

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

$z = \not{?}\ 5$

$m = 5$

Note that = has right to left associativity.

Example 2

```
int main()
{
    int x = 5;
    int y = x, z;
    int m = z = y;

    return 0;
}
```

Tracing on paper:

$x = 5$

$y = 5$

$z = \not{?}\ 5$

$m = 5$

Note that = has right to left associativity.

Example 3

```
int main()
{
    int x = 5;
    int y = x;
    int m = z = y;

    return 0;
}
```

Example 3

```
int main()
{
    int x = 5;
    int y = x;
    int m = z = y;

    return 0;
}
```

Example 3

```
int main()
{
    int x = 5;
    int y = x;
    int m = z = y;

    return 0;
}
```

Example 3

```
int main()
{
    int x = 5;
    int y = x;
    int m = z = y;

    return 0;
}
```

Example 3

```
int main()
{
    int x = 5;
    int y = x;
    int m = z = y;

    return 0;
}
```

Example 3

```
int main()
{
    int x = 5;
    int y = x;
    int m = z = y;

    return 0;
}
```

Remember!

Variables MUST have been declared before first use.

Example 3

```
int main()
{
    int x = 5;
    int y = x;
    int m = z = y; // compile-time error

    return 0;
}
```

Remember!

Variables MUST have been declared before first use.

☺ Thank you! ☺