Introduction to algorithms

The pointers

Mohamed MESSABIHI

mohamed.messabihi@gmail.com

University of Tlemcen
Department of Computer Science
1st year MI

https://sites.google.com/site/informatiquemessabihi/

Introduction

Exercise

Write a function that is passed a duration in minutes. It would return the corresponding number of hours and minutes:

- if we send 45, the function returns 0 hours and 45 minutes;
- if we send 60, the function returns 1 hour and 0 minutes;
- if we send 90, the function returns 1 hour and 30 minutes.

Problems:

- In fact, you can only return one value per function!
- Global variables can be used but this practice is strongly discouraged.

Introduction

Exercise

Write a function that is passed a duration in minutes. It would return the corresponding number of hours and minutes:

- if we send 45, the function returns 0 hours and 45 minutes;
- if we send 60, the function returns 1 hour and 0 minutes;
- if we send 90, the function returns 1 hour and 30 minutes.

Problems:

- In fact, you can only return one value per function!
- Global variables can be used but this practice is strongly discouraged.

Introduction

Exercise

Write a function that is passed a duration in minutes. It would return the corresponding number of hours and minutes:

- if we send 45, the function returns 0 hours and 45 minutes;
- if we send 60, the function returns 1 hour and 0 minutes;
- if we send 90, the function returns 1 hour and 30 minutes.

Problems:

- In fact, you can only return one value per function!
- Global variables can be used but this practice is strongly discouraged.

Introduction

Exercise

Write a function that is passed a duration in minutes. It would return the corresponding number of hours and minutes:

- if we send 45, the function returns 0 hours and 45 minutes;
- if we send 60, the function returns 1 hour and 0 minutes;
- if we send 90, the function returns 1 hour and 30 minutes.

Problems:

- In fact, you can only return one value per function!
- Global variables can be used but this practice is strongly discouraged.

Introduction

Exercise

Write a function that is passed a duration in minutes. It would return the corresponding number of hours and minutes:

- if we send 45, the function returns 0 hours and 45 minutes;
- if we send 60, the function returns 1 hour and 0 minutes;
- if we send 90, the function returns 1 hour and 30 minutes.

Problems:

- In fact, you can only return one value per function!
- Global variables can be used but this practice is strongly discouraged.

Introduction

Exercise

Write a function that is passed a duration in minutes. It would return the corresponding number of hours and minutes:

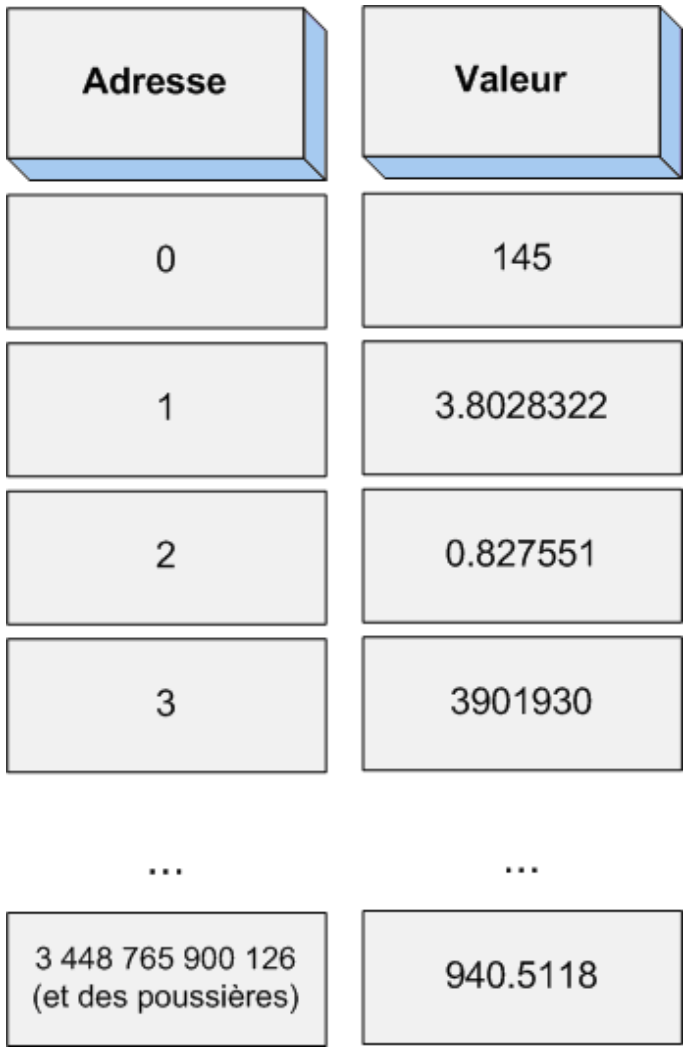
- if we send 45, the function returns 0 hours and 45 minutes;
- if we send 60, the function returns 1 hour and 0 minutes;
- if we send 90, the function returns 1 hour and 30 minutes.

Problems:

- In fact, you can only return one value per function!
- Global variables can be used but this practice is strongly discouraged.

Solution: pointer concept

We must therefore learn to use the concept of a pointer...



Address vs. Value

When you create an age variable of type int for example, by typing:

```
intage= 10;
```

1. Your program asks the operating system (Windows, for example) for permission to use some memory.
2. The operating system responds by indicating at which address in memory it allows you to write your number.
3. The value 10 is written somewhere in memory, say for example at address 4655.
4. Then the compiler replaces the word age in your program with the address 4655 at runtime.
5. So the computer will always go to address 4655 to retrieve the value of the age variable.

Address vs. Value

When you create an age variable of type int for example, by typing:

```
intage= 10;
```

1. Your program asks the operating system (Windows, for example) for permission to use some memory.

2. The operating system responds by indicating at which address in memory it allows you to write your number.

3. The value 10 is written somewhere in memory, say for example at address 4655.

4. Then the compiler replaces the word age in your program with the address 4655 at runtime.

5. So the computer will always go to address 4655 to retrieve the value of the age variable.

Address vs. Value

When you create an age variable of type int for example, by typing:

```
intage= 10;
```

1. Your program asks the operating system (Windows, for example) for permission to use some memory.

2. The operating system responds by indicating at which address in memory it allows you to write your number.

3. The value 10 is written somewhere in memory, say for example at address 4655.

4. Then the compiler replaces the word age in your program with the address 4655 at runtime.

5. So the computer will always go to address 4655 to retrieve the value of the age variable.

Address vs. Value

When you create an age variable of type int for example, by typing:

```
intage= 10;
```

1. Your program asks the operating system (Windows, for example) for permission to use some memory.

2. The operating system responds by indicating at which address in memory it allows you to write your number.

3. The value 10 is written somewhere in memory, say for example at address 4655.

4. Then the compiler replaces the word age in your program with the address 4655 at runtime.

5. So the computer will always go to address 4655 to retrieve the value of the age variable.

Address vs. Value

When you create an age variable of type int for example, by typing:

```
intage= 10;
```

1. Your program asks the operating system (Windows, for example) for permission to use some memory.

2. The operating system responds by indicating at which address in memory it allows you to write your number.

3. The value 10 is written somewhere in memory, say for example at address 4655.

4. Then the compiler replaces the word age in your program with the address 4655 at runtime.

5. So the computer will always go to address 4655 to retrieve the value of the age variable.

Address vs. Value

How to retrieve the address of a variable?

Example :

```
intage= 10;
printf("The age variable is: %d",age);
printf("The address of the age variable is: %p", &age);
```

```
La variable age vaut : 10
L'adresse de la variable age est : 0028FF1C
```

So to remember:

1. age: designates the value of the variable;
2. &age: designates the address of the variable.

Address vs. Value

How to retrieve the address of a variable?

Example :

```
intage= 10;
printf("The age variable is: %d",age);
printf("The address of the age variable is: %p", &age);
```

```
La variable age vaut : 10
L'adresse de la variable age est : 0028FF1C
```

So to remember:

1. age: designates the value of the variable;
2. &age: designates the address of the variable.

Address vs. Value

How to retrieve the address of a variable?

Example :

```
intage= 10;
printf("The age variable is: %d",age);
printf("The address of the age variable is: %p", &age);
```

```
La variable age vaut : 10
L'adresse de la variable age est : 0028FF1C
```

So to remember:

1. age: designates the value of the variable;
2. &age: designates the address of the variable.

What is a pointer?

- So far we have only created variables made to hold numbers.
- Now we will learn how to create variables made to hold addresses
- These are precisely what we call pointers.

Declaring a pointer:

```
int*myPointer;
```

- To create a pointer type variable, we must add the symbol * before the variable name.
- Note that we can also write int* myPointer ;

What is a pointer?

- So far we have only created variables made to hold numbers.
- Now we will learn how to create variables made to hold addresses
- These are precisely what we call pointers.

Declaring a pointer:

```
int*myPointer;
```

- To create a pointer type variable, we must add the symbol * before the variable name.
- Note that we can also write int* myPointer ;

What is a pointer?

- So far we have only created variables made to hold numbers.
- Now we will learn how to create variables made to hold addresses
- These are precisely what we call pointers.

Declaring a pointer:

```
int*myPointer;
```

- To create a pointer type variable, we must add the symbol * before the variable name.
- Note that we can also write int* myPointer ;

What is a pointer?

- So far we have only created variables made to hold numbers.
- Now we will learn how to create variables made to hold addresses
- These are precisely what we call pointers.

Declaring a pointer:

```
int*myPointer;
```

- To create a pointer type variable, we must add the symbol * before the variable name.
- Note that we can also write int* myPointer ;

What is a pointer?

- So far we have only created variables made to hold numbers.
- Now we will learn how to create variables made to hold addresses
- These are precisely what we call pointers.

Declaring a pointer:

```
int*myPointer;
```

- To create a pointer type variable, we must add the symbol * before the variable name.
- Note that we can also write int* myPointer ;

What is a pointer?

- So far we have only created variables made to hold numbers.
- Now we will learn how to create variables made to hold addresses
- These are precisely what we call pointers.

Declaring a pointer:

```
int*myPointer;
```

- To create a pointer type variable, we must add the symbol * before the variable name.
- Note that we can also write int* myPointer ;

Initialize and assign an address to a pointer

- To initialize a pointer, that is to say to give it a default value, we generally do not use the number 0 but the keyword NULL (be sure to write it in capital letters)

Example

```
int*myPointer=NULL;

intage= 10;
int*pointerOnAge= &age;
```

- The first line reserves a slot in memory to hold an address but that slot does not currently contain any addresses.
- The second line means: Create a variable of type int whose value is 10.
- The last line means: Create a pointer type variable whose value is worthaddressof the age variable .

Initialize and assign an address to a pointer

- To initialize a pointer, that is to say to give it a default value, we generally do not use the number 0 but the keyword NULL (be sure to write it in capital letters)

Example

```
int*myPointer=NULL;

intage= 10;
int*pointerOnAge= &age;
```

- The first line reserves a slot in memory to hold an address but that slot does not currently contain any addresses.
- The second line means: Create a variable of type int whose value is 10.
- The last line means: Create a pointer type variable whose value is worthaddressof the age variable .

Initialize and assign an address to a pointer

- To initialize a pointer, that is to say to give it a default value, we generally do not use the number 0 but the keyword NULL (be sure to write it in capital letters)

Example

```
int*myPointer=NULL;

intage= 10;
int*pointerOnAge= &age;
```

- The first line reserves a slot in memory to hold an address but that slot does not currently contain any addresses.
- The second line means: Create a variable of type int whose value is 10.
- The last line means: Create a pointer type variable whose value is worthaddressof the age variable .

Initialize and assign an address to a pointer

- To initialize a pointer, that is to say to give it a default value, we generally do not use the number 0 but the keyword NULL (be sure to write it in capital letters)

Example

```
int*myPointer=NULL;

intage= 10;
int*pointerOnAge= &age;
```

- The first line reserves a slot in memory to hold an address but that slot does not currently contain any addresses.

- The second line means: Create a variable of type int whose value is 10.
- The last line means: Create a pointer type variable whose value is worthaddressof the age variable .

Initialize and assign an address to a pointer

- To initialize a pointer, that is to say to give it a default value, we generally do not use the number 0 but the keyword NULL (be sure to write it in capital letters)

Example

```
int*myPointer=NULL;

intage= 10;
int*pointerOnAge= &age;
```

- The first line reserves a slot in memory to hold an address but that slot does not currently contain any addresses.

- The second line means: Create a variable of type int whose value is 10.
- The last line means: Create a pointer type variable whose value is worthaddressof the age variable .

Do pointers have a type?

- There is no pointer type as there is an int type and a double type. So we do not write pointer pointerToAge ;
- We use the * symbol, but we continue to indicate what is the type of the variable whose pointer will contain the address.

Example :

```
int age = 10;
int *pointerOnAge = &age;
```

- Since the pointer pointerToAge will contain the address of the variable age (which is of type int), then the pointer must be of type int*.
- If the age variable had been of type double, then we would have had to write double *myPointer.

Do pointers have a type?

- There is no pointer type as there is an int type and a double type. So we do not write pointer pointerToAge ;
- We use the * symbol, but we continue to indicate what is the type of the variable whose pointer will contain the address.

Example :

```
intage= 10;
int*pointerOnAge= &age;
```

- Since the pointer pointerToAge will contain the address of the variable age (which is of type int), then the pointer must be of type int*.
- If the age variable had been of type double, then we would have had to write double *myPointer.

Do pointers have a type?

- There is no pointer type as there is an int type and a double type. So we do not write pointer pointerToAge ;
- We use the * symbol, but we continue to indicate what is the type of the variable whose pointer will contain the address.

Example :

```
intage= 10;
int*pointerOnAge= &age;
```

- Since the pointer pointerToAge will contain the address of the variable age (which is of type int), then the pointer must be of type int*.
- If the age variable had been of type double, then we would have had to write double *myPointer.

Do pointers have a type?

- There is no pointer type as there is an int type and a double type. So we do not write pointer pointerToAge ;
- We use the * symbol, but we continue to indicate what is the type of the variable whose pointer will contain the address.

Example :

```
intage= 10;
int*pointerOnAge= &age;
```

- Since the pointer pointerToAge will contain the address of the variable age (which is of type int), then the pointer must be of type int*.
- If the age variable had been of type double, then we would have had to write double *myPointer.

Do pointers have a type?

- There is no pointer type as there is an int type and a double type. So we do not write pointer pointerToAge ;
- We use the * symbol, but we continue to indicate what is the type of the variable whose pointer will contain the address.

Example :

```
intage= 10;
int*pointerOnAge= &age;
```

- Since the pointer pointerToAge will contain the address of the variable age (which is of type int), then the pointer must be of type int*.
- If the age variable had been of type double, then we would have had to write double *myPointer.

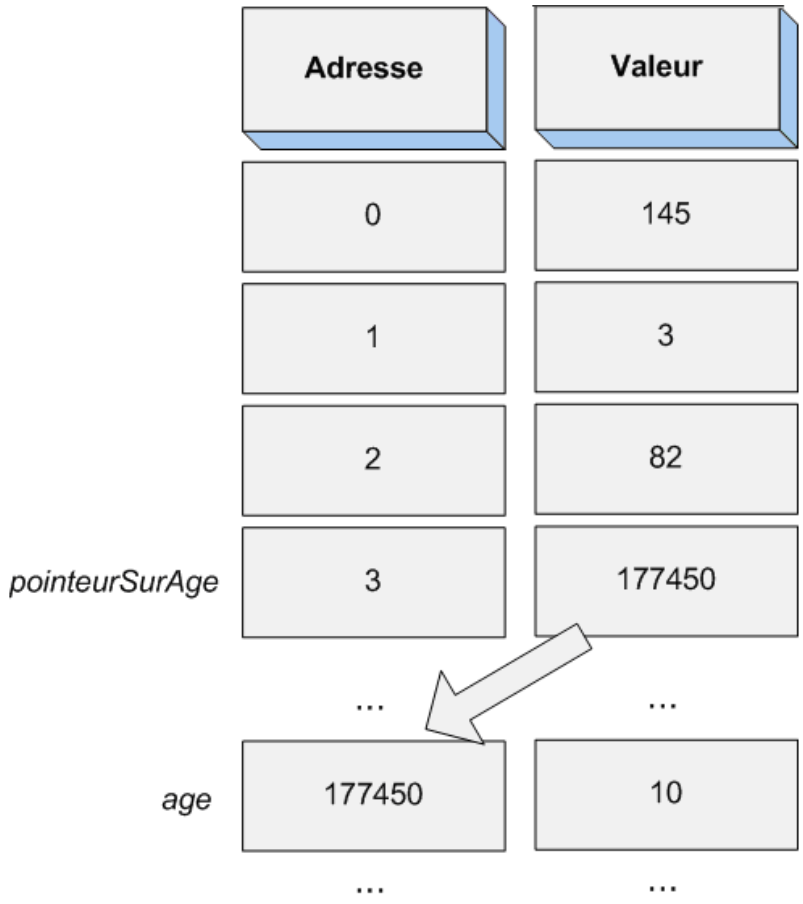
Explanatory diagram

Example :

```
intage= 10;
int*pointerOnAge= &age;
```

Vocabulary :

The pointer is said to pointerOnAge point to the variable age.



The value of a pointer

Example :

```
intage= 10;
int*pointerOnAge= &age;

printf("%d",        pointerOnAge);
printf("%d",        * pointerOnAge);
printf("%d",        &pointerOnAge);
```

- The first printf call displays the value of pointerSurAge, and its value is the address of the age variable (177450).
- The second printf call displays the value of the variable at the address specified in pointerToAge.
- Therefore, you must place the * symbol in front of the pointer name to retrieve the value of the variable located at the address indicated in a pointer.
- The last printf call displays the address where the pointer is located (here it is 3).

The value of a pointer

Example :

```
intage= 10;
int*pointerOnAge= &age;

printf("%d",        pointerOnAge);
printf("%d",        * pointerOnAge);
printf("%d",        &pointerOnAge);
```

- The first printf call displays the value of pointerSurAge, and its value is the address of the age variable (177450).
- The second printf call displays the value of the variable at the address specified in pointerToAge.
- Therefore, you must place the * symbol in front of the pointer name to retrieve the value of the variable located at the address indicated in a pointer.
- The last printf call displays the address where the pointer is located (here it is 3).

The value of a pointer

Example :

```
intage= 10;
int*pointerOnAge= &age;

printf("%d",        pointerOnAge);
printf("%d",        *pointerOnAge);
printf("%d",        &pointerOnAge);
```

- The first printf call displays the value of pointerSurAge, and its value is the address of the age variable (177450).
- The second printf call displays the value of the variable at the address specified in pointerToAge.
- Therefore, you must place the * symbol in front of the pointer name to retrieve the value of the variable located at the address indicated in a pointer.
- The last printf call displays the address where the pointer is located (here it is 3).

The value of a pointer

Example :

```
intage= 10;
int*pointerOnAge= &age;

printf("%d",        pointerOnAge);
printf("%d",        *pointerOnAge);
printf("%d",        &pointerOnAge);
```

- The first printf call displays the value of pointerSurAge, and its value is the address of the age variable (177450).
- The second printf call displays the value of the variable at the address specified in pointerToAge.
- Therefore, you must place the * symbol in front of the pointer name to retrieve the value of the variable located at the address indicated in a pointer.
- The last printf call displays the address where the pointer is located (here it is 3).

The value of a pointer

Example :

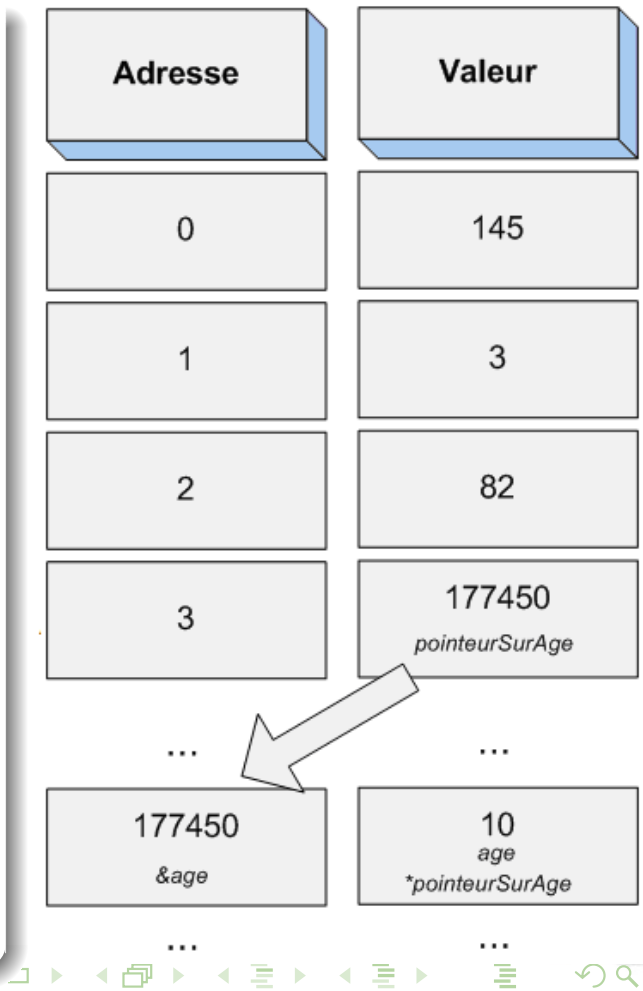
```
intage= 10;
int*pointerOnAge= &age;

printf("%d",        pointerOnAge);
printf("%d",        *pointerOnAge);
printf("%d",        &pointerOnAge);
```

- The first printf call displays the value of pointerSurAge, and its value is the address of the age variable (177450).
- The second printf call displays the value of the variable at the address specified in pointerToAge.
- Therefore, you must place the * symbol in front of the pointer name to retrieve the value of the variable located at the address indicated in a pointer.
- The last printf call displays the address where the pointer is located (here it is 3).

To remember

Pointers and variable names have the same role: They provide access to a location in the computer's internal memory. It is still important to make the difference:

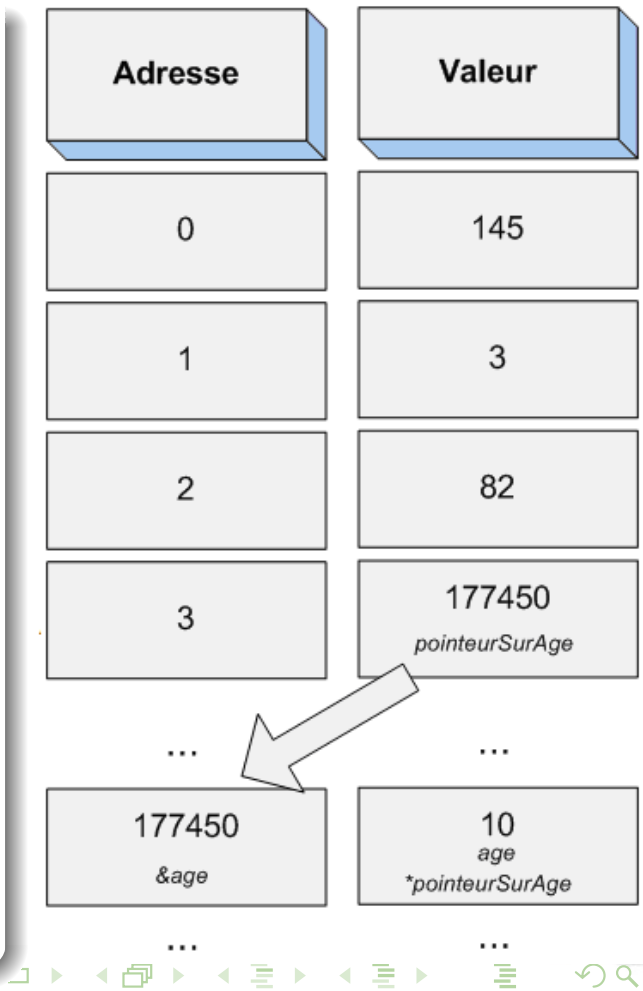
- The name of a variable always remains linked to the same address.
  - agemeans: I want the value of the variable age ,
  - &agemeans: I want the address at which contains the age variable ;
- A pointer is a variable that can pointat different addresses.
  - pointerOnAgemeans: I want it pointer valueSurAge (this value being an address),
  - *pointeurSurAgemeans: Value of the variable located at the address contained in pointeurSurAge



To remember

Pointers and variable names have the same role: They provide access to a location in the computer's internal memory. It is still important to make the difference:

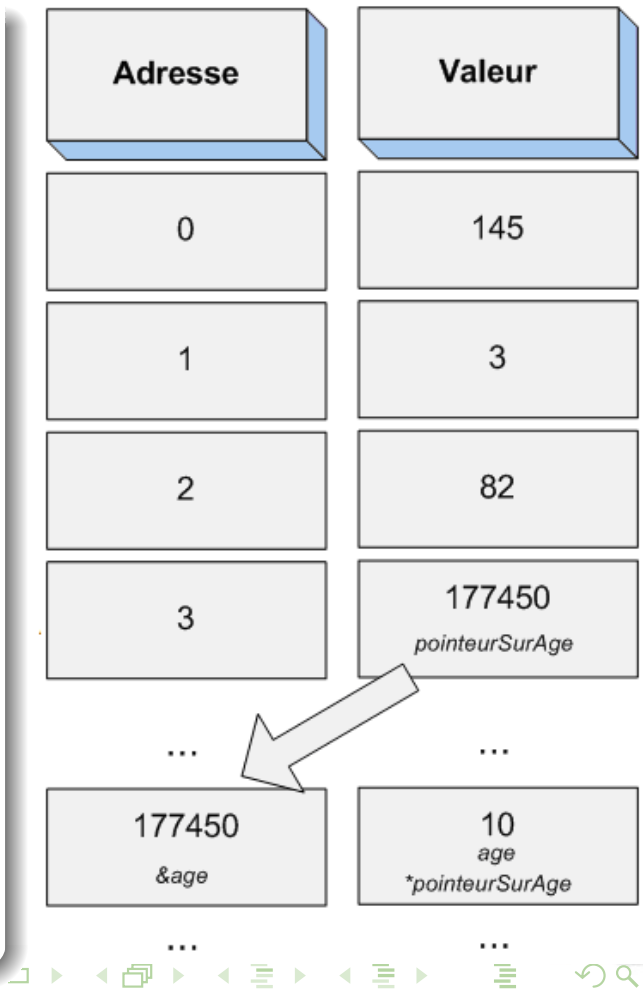
- The name of a variable always remains linked to the same address.
  - agemeans: I want the value of the variable age ,
  - &agemeans: I want the address at which contains the age variable ;
- A pointer is a variable that can pointat different addresses.
  - pointerOnAgemeans: I want it pointer valueSurAge (this value being an address),
  - *pointerOnAgemeans: valueI want it variable located at the address contained in pointerSurAge .



To remember

Pointers and variable names have the same role: They provide access to a location in the computer's internal memory. It is still important to make the difference:

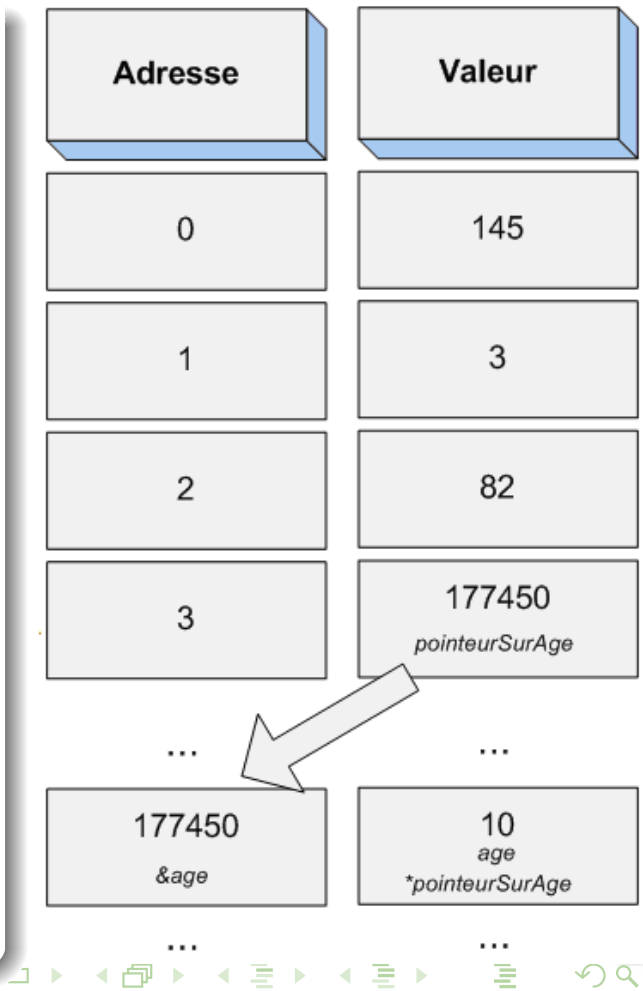
- The name of a variable always remains linked to the same address.
  - agemeans: I want the value of the variable age ,
  - &agemeans: I want the address at which contains the age variable ;
- A pointer is a variable that can pointat different addresses.
  - pointerOnAgemeans: I want it pointer valueSurAge (this value being an address),
  - *pointerOnAgemeans: valueI want it variable located at the address contained in pointerSurAge .



To remember

Pointers and variable names have the same role: They provide access to a location in the computer's internal memory. It is still important to make the difference:

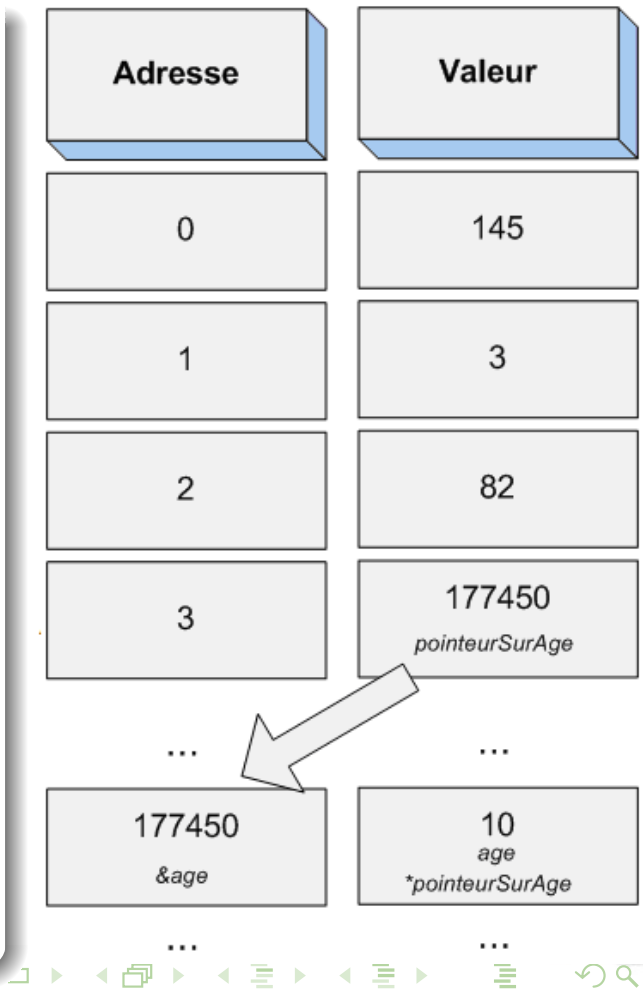
- The name of a variable always remains linked to the same address.
  - agemeans: I want the value of the variable age ,
  - &agemeans: I want the address at which contains the age variable ;
- A pointer is a variable that can pointat different addresses.
  - pointerOnAgemeans: I want it pointer valueSurAge (this value being an address),
  - *pointerOnAgemeans: value I want it variable located at the address contained in pointerSurAge .



To remember

Pointers and variable names have the same role: They provide access to a location in the computer's internal memory. It is still important to make the difference:

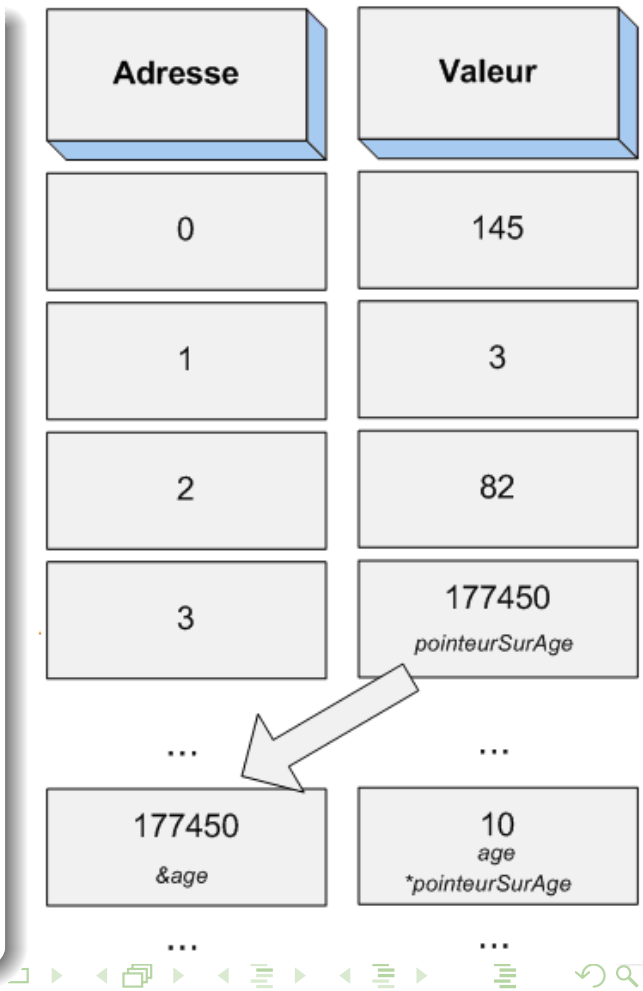
- The name of a variable always remains linked to the same address.
  - agemeans: I want the value of the variable age ,
  - &agemeans: I want the address at which contains the age variable ;
- A pointer is a variable that can pointat different addresses.
  - pointerOnAgemeans: I want it pointer valueSurAge (this value being an address),
  - *pointerOnAgemeans: value of the variable located at the address contained in pointerSurAge .



To remember

Pointers and variable names have the same role: They provide access to a location in the computer's internal memory. It is still important to make the difference:

- The name of a variable always remains linked to the same address.
  - agemeans: I want the value of the variable age ,
  - &agemeans: I want the address at which contains the age variable ;
- A pointer is a variable that can pointat different addresses.
  - pointerOnAgemeans: I want it pointer valueSurAge (this value being an address),
  - *pointerOnAgemeans: I want the value of the variable located at the address contained in pointerSurAge .



Declaration vs. Using a Pointer

Attention :

Do not confuse the different meanings of the star!

1. When wedeclareda pointer, the star is just used to indicate that we want to create a pointer: int *pointerOnAge;.
2. On the other hand, when we thenusedthe pointer by writing printf("%d", *pointerOnAge);, this does not mean we want to create a pointer but: we want the value of the variable to which the pointerOnAge points.

Example :

```
intage= 10;
int*pointerOnAge= &age;                    // declaration of a pointer

* pointerOnAge+= 1; printf("%d", *         // use          of   pointer
pointerOnAge);                             // use          of   pointer
```

Passing parameters of a function by address

The big advantage of pointers (but it is not the only one) is that we can send them to functions so that they directly modify a variable in memory, and not a copy as we have already seen before.

Example :

```
void    triplePointer(int*pointerToNumber); hand()
int
{
        intnumber= 5;

        triplePointer(&number);// We send the address of
                number has the function
        printf("%d",number);// We display the number variable.

        return0;
}
void    triplePointer(int               * pointerToNumber)
{
        * pointerToNumber           * = 3;
}
```

Passing parameters by address functions

By running the program from the previous slide, here is what happens in order, starting from the beginning of main:

1. a variable `number` is created in the main. It is given the value 5.
2. we call the function `triplePointer`. We send it as a parameter the address of our number variable;
3. the function `triplePointer` receives this address in `pointerToNumber`. Inside the function `triplePointer`, so we have a pointer `pointerToNumber` which contains the address of the variable `number`;
4. now that we have a pointer to `number`, we can directly modify the variable `number` in memory! It is enough to use `* pointerToNumber` to designate the variable `number`! For the example, we simply multiply the variable `number` by 3;
5. back in the main function, our `number` is now worth 15 because the function `triplePointer` directly modified the value of `number`.

Passing parameters by address functions

By running the program from the previous slide, here is what happens in order, starting from the beginning of main:

1. a variable `number` is created in the main. It is given the value 5.
2. we call the function `triplePointer`. We send it as a parameter the address of our number variable;
3. the function `triplePointer` receives this address in `pointerToNumber`. Inside the function `triplePointer`, so we have a pointer `pointerToNumber` which contains the address of the variable `number`;
4. now that we have a pointer to `number`, we can directly modify the variable `number` in memory! It is enough to use `* pointerToNumber` to designate the variable `number`! For the example, we simply multiply the variable `number` by 3;
5. back in the main function, our `number` is now worth 15 because the function `triplePointer` directly modified the value of `number`.

Passing parameters by address functions

By running the program from the previous slide, here is what happens in order, starting from the beginning of main:

1. a variable number is created in the main. It is given the value 5.
2. we call the function triplePointer. We send it as a parameter the address of our number variable;
3. the function triplePointer receives this address in pointerToNumber. Inside the function triplePointer, so we have a pointer pointerToNumber which contains the address of the variable number;
4. now that we have a pointer to number, we can directly modify the variable number in memory! It is enough to use * pointerToNumber to designate the variable number! For the example, we simply multiply the variable number by 3;
5. back in the main function, our number is now worth 15 because the function triplePointer directly modified the value of number.

Passing parameters by address functions

By running the program from the previous slide, here is what happens in order, starting from the beginning of main:

1. a variable number is created in the main. It is given the value 5.
2. we call the function triplePointer. We send it as a parameter the address of our number variable;
3. the function triplePointer receives this address in pointerToNumber. Inside the function triplePointer, so we have a pointer pointerToNumber which contains the address of the variable number;
4. now that we have a pointer to number, we can directly modify the variable number in memory! It is enough to use * pointerToNumber to designate the variable number! For the example, we simply multiply the variable number by 3;
5. back in the main function, our number is now worth 15 because the function triplePointer directly modified the value of number.

Passing parameters by address functions

By running the program from the previous slide, here is what happens in order, starting from the beginning of main:

1. a variable `number` is created in the main. It is given the value 5.
2. we call the function `triplePointer`. We send it as a parameter the address of our number variable;
3. the function `triplePointer` receives this address in `pointerToNumber`. Inside the function `triplePointer`, so we have a pointer `pointerToNumber` which contains the address of the variable `number`;
4. now that we have a pointer to `number`, we can directly modify the variable `number` in memory! It is enough to use `* pointerToNumber` to designate the variable `number`! For the example, we simply multiply the variable `number` by 3;
5. back in the main function, our `number` is now worth 15 because the function `triplePointer` directly modified the value of `number`.

Passing parameters by address functions

By running the program from the previous slide, here is what happens in order, starting from the beginning of main:

1. a variable number is created in the main. It is given the value 5.
2. we call the function triplePointer. We send it as a parameter the address of our number variable;
3. the function triplePointer receives this address in pointerToNumber. Inside the function triplePointer, so we have a pointer pointerToNumber which contains the address of the variable number;
4. now that we have a pointer to number, we can directly modify the variable number in memory! It is enough to use * pointerToNumber to designate the variable number! For the example, we simply multiply the variable number by 3;
5. back in the main function, our number is now worth 15 because the function triplePointer directly modified the value of number.

What if we went back to our initial exercise?

Solution

```
void    cutMinutes(int,int*pointerHours,int* pointerMinutes);

int    hand()
{
        intduration=0;
        inthours= 0,minutes=0 ;
        printf("Give the number of minutes: \n");scanf("%d",
                &duration);
        // We send the address of hours and minutes cutMinutes(duration, &
        hours, &minutes); // This time the values   have been changed! printf("%d
        hours and %d minutes",hours,minutes); return0;


}
void    cutMinutes(intduration,int*pointerHours,int* pointerMinutes)

{
        *pointerHours=duration/ 60;
        *pointerMinutes=duration% 60;
}
```

In summary

- Each variable is stored at a specific address in memory.
- Pointers are similar to variables. Instead of storing a number they store the address at which a variable is located in memory.
- If we place a & symbol in front of a variable name, we get its address instead of its value (eg: & age).
- Placing a * symbol in front of a pointer name gives the value of the variable stored at the address indicated by the pointer.
- Pointers are an essential concept of the C language, but nevertheless a little complex at first. It is necessary to take the time to understand how they work because many other concepts are based on them.

In summary

- Each variable is stored at a specific address in memory.
- Pointers are similar to variables. Instead of storing a number they store the address at which a variable is located in memory.
- If we place a & symbol in front of a variable name, we get its address instead of its value (eg: & age).
- Placing a * symbol in front of a pointer name gives the value of the variable stored at the address indicated by the pointer.
- Pointers are an essential concept of the C language, but nevertheless a little complex at first. It is necessary to take the time to understand how they work because many other concepts are based on them.

In summary

- Each variable is stored at a specific address in memory.
- Pointers are similar to variables. Instead of storing a number they store the address at which a variable is located in memory.
- If we place a & symbol in front of a variable name, we get its address instead of its value (eg: & age).
- Placing a * symbol in front of a pointer name gives the value of the variable stored at the address indicated by the pointer.
- Pointers are an essential concept of the C language, but nevertheless a little complex at first. It is necessary to take the time to understand how they work because many other concepts are based on them.

In summary

- Each variable is stored at a specific address in memory.
- Pointers are similar to variables. Instead of storing a number they store the address at which a variable is located in memory.
- If we place a & symbol in front of a variable name, we get its address instead of its value (eg: & age).
- Placing a * symbol in front of a pointer name gives the value of the variable stored at the address indicated by the pointer.
- Pointers are an essential concept of the C language, but nevertheless a little complex at first. It is necessary to take the time to understand how they work because many other concepts are based on them.

In summary

- Each variable is stored at a specific address in memory.
- Pointers are similar to variables. Instead of storing a number they store the address at which a variable is located in memory.
- If we place a & symbol in front of a variable name, we get its address instead of its value (eg: & age).
- Placing a * symbol in front of a pointer name gives the value of the variable stored at the address indicated by the pointer.
- Pointers are an essential concept of the C language, but nevertheless a little complex at first. It is necessary to take the time to understand how they work because many other concepts are based on them.