• **int**: Basic signed integer type.
• **unsigned int**: Basic unsigned (only positive) integer type.
• **float**: Real number, usually referred to as a single-precision floating-point type.
• **double**: Real number, usually referred to as a double-precision floating-point type.
• **char**: Character type.
• ...
type name;
  type name = value;
  const type name = value;

- **type**: The data type (see previous slide).
- **name**: The name of the variable.
  - Only lowercase, uppercase and digits (accents are forbidden);
  - Must start with a letter;
  - Spaces are prohibited;
- **value**: Value given to the variable.
- **const**: To define a constant number (cannot change after its definition).
Listing 1: age.c

```c
if(age < 12) // If x is smaller than 12
{
    printf("You are a child.\n");
}
else if(age >= 12 && age < 18) // If x is greater or
//equal to 12 and smaller than 18
{
    printf("You are a teenager.\n");
}
else // Else (if x is greater or equal to 18)
{
    printf("Your are an adult.\n");
}
```
Listing 2: number.c

```c
switch (x)
{
    case 0:
        printf("x = 0\n");
        break;
    case 1:
        printf("x = 1\n");
        break;
    case 2:
        printf("x = 2\n");
        break;
    default:
        printf("x is different from 0, 1 and 2.\n");
        break;
}
```
Listing 3: factorial.c

```c
int factorial = 1;

for(int i = 1; i <= 10; i++)
{
    factorial *= i;
}

printf("Factorial of 10 = %d \n", factorial);
```
Listing 4: sum.c

```c
int sum = 0;

while (sum < 100)
{
    sum ++;
}

printf("sum = %d \n", sum);
```
Introducing Pointers

### Memory Layout

<table>
<thead>
<tr>
<th>Memory</th>
<th>a</th>
<th>10</th>
<th>3.9</th>
<th>...</th>
<th>0xff4aed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>0xff4aec</td>
<td>0xff4aed</td>
<td>0xff4aee</td>
<td>...</td>
<td>0xffa870</td>
</tr>
</tbody>
</table>

### Listing 5: pointers.c

```c
int number = 10;
int *pointer = &number;

printf("The value of number is: \%d \n", number);
printf("The address of number is: \%p \n\n", &number);

printf("The value of pointer is: \%p \n", pointer);
printf("The address of pointer is: \%p \n", &pointer);
printf("The value pointed to by pointer is: \%d \n", *pointer);
```

---

**Note:** The memory addresses are for illustrative purposes and may not correspond to actual memory allocation.
Listing 6: static_array.c

```c
char primeNumber[6] = {2, 3, 5, 7, 11, 13};

printf("The 6 first prime number are: \n");
for(int i = 0; i < 6; i++)
{
    printf("%d \n", primeNumber[i]);
}
```
Listing 7: dynamic_array.c

```c
int arrayOfNumber[7] = {1, 3, 5, 3, 4, 7, 3};
int countThree = 0;

// We use a loop to count the number of 3
for(int i = 0; i < 7; i++)
{
    if(arrayOfNumber[i] == 3)
    {
        countThree ++;
    }
}

// We create a array of size countThree containing data of type int
int *threeIndex = malloc(countThree*sizeof(int));
```
Dynamic array

// We store indexes in the array threeIndex
int numThree = 0;
for(int i = 0; i < 27; i++)
{
    if(arrayOfNumber[i] == 3)
    {
        threeIndex[numThree] = i;
        numThree ++;
    }
}

printf("There are %d number three located at: [", countThree);
for(int i = 0; i < countThree; i++)
{
    printf("%d ", threeIndex[i]);
}
printf("] \n");

// Don’t forget to free the memory!!
free(threeIndex);
In C language a string:

- Is seen as an array of characters (char);
- **Must be finished** by a special character '\0' (backslash and zero).

Listing 8: hello.c

```c
char hello[] = "Hello";

for (int i = 0; i < 6; i++)
{
    printf("%c \n", hello[i]);
}
```
Listing 9: string.c

char name[] = "C. Geuzaine";
char professor[] = "Professor ";

//size of a string: function strlen(str)
printf("Size of name = %ld \n", strlen(name));

char bigString[50];

//copy a string: function strcpy(str1, str2)
strcpy(bigString, professor);

//concatenates 2 strings: function strcat(str1, str2)
strcat(bigString, name);
printf("%s \n", bigString);
You can also convert a string to numerical value (int, float, double, ...):

```c
char myInteger[] = "45";
char myReal[] = "3.14";

int numI = atoi(myInteger);
double numD = atof(myReal);

printf("numI %d\n", numI);
printf("numD %lf\n\n", numD);
```
Main function

All programs must contain a main function. This is the start point of a program.

There are two parameters:
- \texttt{argc}: The number of parameters passed to the program;
- \texttt{argv}: The value of these parameters;

There is always one parameter that contains the name of the program (in \texttt{argv[0]}).

Listing 10: parameters.c

```c
int main(int argc, char* argv[])
{
    printf("There are %d parameters: \n", argc);
    for(int i = 0; i < argc; i++)
    {
        printf("%s \n", argv[i]);
    }
    printf("\n");

    return 0;
}
```
Main function

If you compile and then run the previous code with command:

```
./myProgram abcd toto 3.1415
```

you obtain,

There are 4 parameters:
```
./myProgram
abcd
toto
3.1415
```
A function:

- Must be declared before it is used;
- Has a name;
- Has a return type (or `void` if there is no return);
- Has parameters;

Example:

```c
char myFunction(int param1, double param2);
```

`myFunction` takes two parameters and returns a `char` value. The first parameter is named `param1` of type `int` and the second parameter is named `param1` of type `double`. 
Listing 11: function.c

// The function must be declared before it is used
int square(int number);

int main(int argc, char** argv)
{
    int number = 3;

    // The use of the function
    printf("The square of %d is %d\n", number, square(number));

    return 0;
}

// The implementation of the function
int square(int number)
{
    return number*number;
}
Listing 12: functionAndPointers.c

```c
void myFunction1(int number);
void myFunction2(int* pointerOnNumber);

int main(int argc, char** argv)
{
    int number = 3;

    myFunction1(number);
    printf("number = %d\n", number);

    myFunction2(&number);
    printf("number = %d\n", number);

    return 0;
}
```
Functions and pointers

```c
void myFunction1(int number) {
    number = number + 1;
}

void myFunction2(int* pointerOnNumber) {
    *pointerOnNumber = *pointerOnNumber + 1;
}
```
To write on the screen:

```c
printf("A text");
printf("A text with a int %d", num1);
printf("A text with a char %c and a double %lf", num2, num3);
```

To read from the screen:

```c
//read an integer (int myInteger)
scanf("%d", &myInteger);
//read an character (char myChar)
scanf("%c", &myChar);
//read an float number (float myFloat)
scanf("%f", &myFloat);
```
Listing 13: io.c

```c
int number;
char character;

printf("Please enter a character: ");
scanf("%c", &character);

printf("You have enter %c \n", character);

printf("Please enter an integer number: ");
scanf("%d", &number);

printf("You have enter %d \n", number);
```

Listing 14: file.c

FILE *fileW;
// Open the file for writing
fileW = fopen("file.txt","w");

// Check if the file is open
if(fileW == NULL)
{
    // If not exit
    return 1;
}

char string[] = "Something the write in the file\n";

for(int i = 0; i < strlen(string); i++)
{
    fputc(string[i], fileW);
}

// Close the file
fclose(fileW);
FILE *fileR;
// Open the file for reading
fileR = fopen("file.txt","r");

// Check if the file is open
if(fileR == NULL)
{
    // If not exit
    return 1;
}

char character = 0;
while(character != '\n')
{
    character = fgetc(fileR);
    printf("%c", character);
}

// Close the file
fclose(fileR);
FILE *file;
//Open the file for reading
file = fopen("HPC.txt","r");

//Check if the file is open
if(file == NULL)
{
    //if not exit
    return 1;
}

char str[100];
int numE = 0;
while(fgets(str, 100, file) != NULL)
{
    for(int i = 0; i < strlen(str); i++)
if(str[i] == 'e')
{
    numE ++;
}
}

printf("There are %d e in this text. \n", numE);

//Close the file
fclose(file);
Environmental variables

You can defined variables in your computer (outside the program). These variables are called **environmental variables**. For example PATH is an environmental variable defined on every computer. If you want to define an environmental variable called `MY_VARIABLE` and set this variable to 3, you can write in command line:

```bash
export MY_VARIABLE=3
```
Then, the following program print the value of this variable (3 in this case):

Listing 16: env.c

```c
char *numThreads = getenv("MY_VARIABLE");

// If the environmental variable don’t exist
if (numThreads == NULL)
{
    return 1;
}

int num = atoi(numThreads);

printf("Environmental variable MY_VARIABLE = %i
\n", num);
```
• https://www.cprogramming.com/tutorial.html
• https://en.wikibooks.org/wiki/C_Programming
• https://openclassrooms.com/courses
• On NIC4 you can copy the previous examples:
  cp -r /home/ulg/ace/aroyer myDir