Informatics 3.
Lecture 2: Basics of C

Kristóf Kovács

Budapest University of Technology and Economics

2024-02-20

Variables

- Declaration

```
int x;
```

Variables

- Declaration

```
int x;
```

- Definition

```
int x = 5;
```

Variables

- Declaration

```
int x;
```

- Definition

```
int x = 5;
```

- Changing the value

```
x = 5;
```

Variables

- Declaration

```
int x;
```

- Definition

```
int x = 5;
```

- Changing the value

```
x = 5;
```

- We need to declare or define a variable before we can use it.

Variables

- Declaration

```
int x;
```

- Definition

```
int x = 5;
```

- Changing the value

```
x = 5;
```

- We need to declare or define a variable before we can use it.
- After a declaration the value of the variable will be undefined:

```
int x;
printf("%d", x);
...
1147283347
```

Functions

- Function declaration

```
float rectangle(float a, float b);
```

Functions

- Function declaration

```
float rectangle(float a, float b);
```

- Function definition

```
float rectangle(float a, float b) {
    return a * b;
}
```

Functions

- Function declaration

```
float rectangle(float a, float b);
```

- Function definition

```
float rectangle(float a, float b) {
    return a * b;
}
```

- A function can be used if it has at least been declared before:

```
float rectangle(float a, float b);

int main(void) {
    printf("%f", rectangle(5, 7));
    return 0;
}
float rectangle(float a, float b) {
    return a * b;
}
```

Function parameters

- The arguments of a function are only copies:

```
void wrong(float x, float y, float sum) {
    sum = x + y;
}

int main(void) {
    float a = 0.0;
    wrong(5.0, 2.0, a);
    printf("%f", a);
    return 0;
}
...
0.0
```

Input function

- Write a function that reads a variable from the user and returns it

Input function

- Write a function that reads a variable from the user and returns it:

```
int input() {
    int n;
    printf("Please input an integer number: ");
    scanf("%d", &n);
    return n;
}

int main(void) {
    int a = input();
    printf("The square of %d is %d", a, a * a);
    return 0;
}
```

struct

- Write a function that reads two numbers from the user

struct

- Write a function that reads two numbers from the user:

```
struct two {
    int a;
    int b;
};
struct two input() {
    int a,b;
    scanf("%d", &a);
    scanf("%d", &b);
    struct two k;
    k.a = a;
    k.b = b;
    return k;
}
```

struct

- Write a function that reads two numbers from the user:

```
struct two {
    int a;
    int b;
};
struct two input() {
    struct two k;
    scanf("%d", &(k.a));
    scanf("%d", &(k.b));
    return k;
}
int main() {
    struct two s = input();
    printf("%d, %d", s.a, s.b);
    return 0;
}
```

typedef

- You can rename a type with the *typedef* keyword:

```
struct two {
    int a, b;
};
typedef struct two rec;
rec input() {
    rec k;
    scanf("%d", &(k.a));
    scanf("%d", &(k.b));
    return k;
}
int main() {
    rec s = input();
    printf("%d, %d", s.a, s.b);
    return 0;
}
```

typedef

- You can rename a type with the *typedef* keyword:

```
typedef struct two {
    int a, b;
} rec;
rec input() {
    rec k;
    scanf("%d", &(k.a));
    scanf("%d", &(k.b));
    return k;
}
int main() {
    rec s = input();
    printf("%d, %d", s.a, s.b);
    return 0;
}
```

Practice

- Define a data structure that can be used to identify a person. Let's suppose we have a *string* type.

Practice

- Define a data structure that can be used to identify a person. Let's suppose we have a *string* type.
  - Name
  - Mother's name
  - Date of birth
  - City of birth

Pointers

- Why is there a random value in a variable if we only declared it?

Pointers

- Why is there a random value in a variable if we only declared it?
- Every variable is stored in your computer's memory

Pointers

- Why is there a random value in a variable if we only declared it?
- Every variable is stored in your computer's memory
- The random value was the slice of a previously stored variable.

Pointers

- Why is there a random value in a variable if we only declared it?
- Every variable is stored in your computer's memory
- The random value was the slice of a previously stored variable.
- You can get the address of a variable in memory, this is called its *pointer*.

Pointers

- Why is there a random value in a variable if we only declared it?
- Every variable is stored in your computer's memory
- The random value was the slice of a previously stored variable.
- You can get the address of a variable in memory, this is called its *pointer*.

```
int main() {
 int x = 25;
 int *x_p = &x;
 printf("The variable stored at the address %p is %d.",
   x_p, *x_p);
 return 0;
}
```

```
...
The variable stored at the address 0x7ffd97aeb0fc is 25.
```

Pointers 2

- The two main pointer operators:
  - &

```
int x = 25;
int *x_p = &x;
```

  - *

```
int x = 25;
int *x_p = &x;
int y = *x_p;
```

Pointers 2

- The two main pointer operators:
  - &

```
int x = 25;
int *x_p = &x;
```

  - *

```
int x = 25;
int *x_p = &x;
int y = *x_p;
```

- You can also get the pointer of a pointer (and so on):

```
int x = 25;
int *x_p = &x;
int **x_pp = &x_p;
int y = **x_pp;
```

Input with pointers

```c
void input(int *a, int *b) {
    scanf("%d", a);
    scanf("%d", b);
}

int main() {
    int a, b;
    input(&a, &b);
    printf("%d, %d", a, b);
    return 0;
}
```

Practice

- Write a function that calculates the area and the perimeter of a rectangle given the length of its sides. You have to use a function with a *void* return type.

Practice

- Write a function that calculates the area and the perimeter of a rectangle given the length of its sides. You have to use a function with a *void* return type.
- Write a function that calculates the square of an integer number in place. The parameter is the pointer of the integer number.

Arrays

- You can think of an array as a python list with limitations

Arrays

- You can think of an array as a python list with limitations:

```
int t[3];
t[0] = 1;
t[1] = 2;
t[2] = 5;
```

Arrays

- You can think of an array as a python list with limitations:

```
int t[3];
t[0] = 1;
t[1] = 2;
t[2] = 5;
```

- You can also do this, but only during definition:

```
int t[] = {1, 2, 5};
```

Arrays

- You can think of an array as a python list with limitations:

```
int t[3];
t[0] = 1;
t[1] = 2;
t[2] = 5;
```

- You can also do this, but only during definition:

```
int t[] = {1, 2, 5};
```

- Arrays have fixed length that has to be declared upon its creation.

Input with arrays

- Why does this work?

```
void input(int t[]) {
    scanf("%d", &t[0]);
    scanf("%d", &t[1]);
    scanf("%d", &t[2]);
}

int main() {
    int t[3];
    input(t);
    printf("%d, %d, %d", t[0], t[1], t[2]);
    return 0;
}
```

Array as pointer

- Arrays are pointers, pointers are arrays

Array as pointer

- Arrays are pointers, pointers are arrays:

```
int main() {
    int t[] = {1, 2, 5};
    int x = 3;
    int *x_p = &x;
    printf("First element of the array: %d.\n", t[0]);
    printf("Array as pointer: %d.\n", *t);
    printf("Pointer as array: %d.\n", x_p[0]);
    return 0;
}
```

```
...
First element of the array: 1.
Array as pointer: 1.
Pointer as array: 3.
```

Array as pointer 2

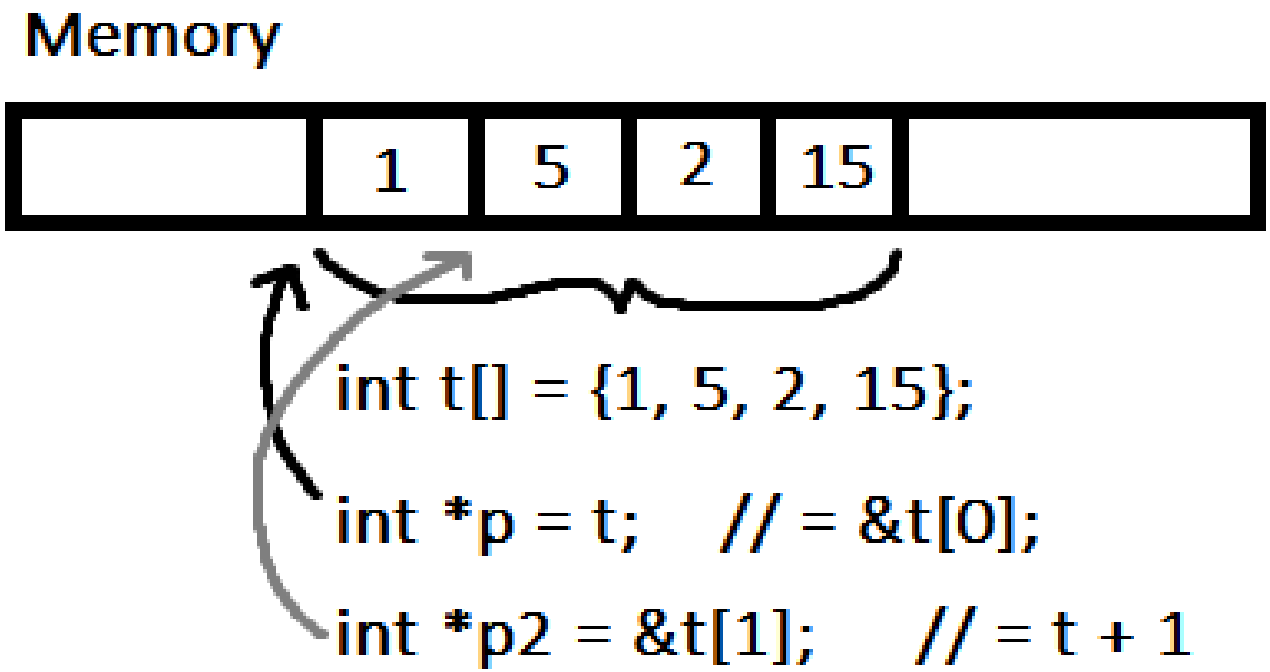
- You can add integer numbers to pointers. This is similar to array indices:

```
int t[] = {1, 5, 2, 15};
int *p = t + 1;
int x = *(t + 1);      // 5
```

Array as pointer 2

- You can add integer numbers to pointers. This is similar to array indices:

```
int t[] = {1, 5, 2, 15};
int *p = t + 1;
int x = *(t + 1);       // 5
```



Practice

- Write a function that reverses a given array.

Practice

- Write a function that reverses a given array.
- Write a program that reads an integer between 1 and 12 from the user. The program outputs the given month of the year (1 = January...).

Practice

- Write a function that reverses a given array.
- Write a program that reads an integer between 1 and 12 from the user. The program outputs the given month of the year (1 = January...).
- Write a function that applies the previously written square function to all elements of a given array.

Pop quiz questions

- Define a *float* array with 2.2, 5.4, 1.4 elements.
- Write a function that outputs the square of the number given through a parameter.
- Define an integer variable and its pointer.
- Define a *struct* type that stores 3 *float* types.
- Give an example of a variable definition and a declaration.