Mechatronic Engineering

Object Oriented Programing and Software Engineering
Laboratory instruction 3
C++ introduction

AGH Kraków, 2020

Materials created for educational purposes.
Dedicated for students attending Software Engineering course.
Author would apreaciate any feedback regarding errors of any kind found in the instruction script.
Please report those to the following email address: danielt@agh.edu.pl

Spis treści

1   Pointers.

A pointer is an object that stores the address (memory) of the variable assigned to it.

Example:

```cpp
#include <iostream>
using namespace std;

main() {
    int a;
    int *b;
    a = 11;
    cout << "a = " << a;
    cout << endl;
    b = &a; /in this line a pointer is connected to a variable
        address*/
    cout <<" value of a variable pointed by b is equal "
    << *b;
    cout << endl;
    *b = 14;
    cout << "a = " << a;
    cout << endl;
return 0;
}
```

Pointers can be used for:
– improved work with arrays,
– functions that can change the value of arguments passed to them,
– access to special memory cells,
– reservation of memory areas.

1.1   VOID type pointers.

Pointers as mentioned above are used to indicate the address of an object in memory. When defining a pointer, keep in mind that its type corresponds to the type of object pointed. You can understand that by creating a pointer to an object we have knowledge about its type.

By using the void indicator we consciously give up this "knowledge" and our pointer automatically loses the ability to read the space it points to. It is possible to "recover" this functionality by using casting:

```
int a;
void *wsk;
(int*)wsk = &a;
```

This impaired indicator can be used to point to other indicators. If you want a pointer of the specified type to point to a void pointer, you must cast.
If you want to display the value of a variable or pointer to which the void pointer indicates, you must also cast the type.

Example:

```
#include <iostream>
using namespace std;

main() {
    int masa=135, *wag, wyb=0;
    float km = 14.96, *moc;
    void *wsk;
    moc = &km;
    wag = &masa;
    cout << "Dobierz motocykl o pojemnosci do 125cm^3 dla siebie\n"
    << "Jaki parametr jest dla Ciebie najistotniejszy?\n"
    << "\t1. Masa\n"
    << "\t2. Moc silnika\n"
    << "Wprowadz swoj wybor: ";
    cin >> wyb;
    if (wyb == 1) {
       wsk = wag;
       cout << "Motocykl dla Ciebie to: Yamacha Virago 125\n"
       <<"Jego masa to: " << *(int*)wsk << "Kg" << endl;
    }
    if (wyb == 2) {
       wsk = moc;
       cout << "Motocykl dla Ciebie to: Honda VT Shadow 125\n"
       <<"Jego moc to: " << *(float*)wsk << "KM" << endl;
    }
return 0;
}
```

1.2 Pointers and arrays.

Suppose we have the following situation:

```cpp
int *wsk;
int tab[5];
wsk=&tab[n];
```

This way, we assign a wsk pointer to the nth tab element. If you want to assign a pointer to a zero element, just type the name of the array (the name of the array always points to its zero element).

```cpp
wsk=&tab;
```

If during program operation we want the pointer to point to another element of this array, just use the following statement:

```cpp
wsk = wsk+1;
//lub
wsk++;
```

If during program operation we want the pointer to point to another element of this array, just use the following statement:

```cpp
wsk += n;
```

For types other than int we do not need to worry that the above method will jump to the next bit. Because the compiler knows what type of pointer we have, it knows how many bits must move to find the next element in the array (Figure 1).

Example:

```cpp
#include <iostream>
using namespace std;

main() {
    float tab[4] = {7.23, 12.46, 16.12, 23.28};
    float *wsk;
    wsk=tab;
    cout << "Najblizszy pociag z Krakowa do "
    << "Krainymlekiemimiodemplynacej odjezdza o: "
    << *wsk << endl;
    wsk=wsk+1;
    cout << "Nastepny pociag odjezdza o: " << *wsk << endl;
    wsk++;
```
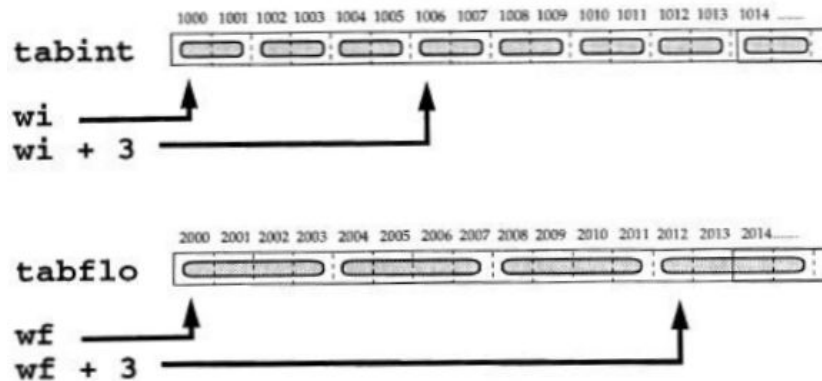
Rysunek 1: Movement through the contents of the array with a pointer. Source: Symfonia C++ J. Grębosz.

```cpp
    cout << "Kolejny pociag odjezdza o: " << *wsk << endl;
    wsk+=1;
    cout << "Ostatni pociag odjezdza o: " << *wsk << endl;
return 0;
}
```

1.3 Pointers and functions

Pointers as function arguments are used to change an object's value by a function. From previous laboratories we know that the easiest way to do this is to use the function of the desired type and return a value. Unfortunately, this method allows us to return only one value. In situations where we want the function to affect more than one object, pointers are necessary.

Example:

```cpp
#include <iostream>
using namespace std;

void pizza(int *lpi, float *gotowka);

main() {
    int liczba_pizzy = 0;
    float pln = 134.28;
    cout << "Obecnie posiadasz " <<liczba_pizzy << " pizze.\n";
    cout << "Masz przy sobie " << pln << " PLN." << endl;
    cout << "Czas zamowic pizze!" << endl;
    pizza(&liczba_pizzy, &pln);
```

```
    cout << "Twoja dostawa nadeszla\n";
    cout << "Teraz posiadasz " << liczba_pizzy << " pizze.\n";
    cout << "Zostalo Ci " << pln << " PLN." << endl;
return 0;
}

void pizza (int *lpi, float *gotowka) {
    *lpi = 2;
    *gotowka = 62.47;
}
```

In the above code, we managed to change the values of two variables with one function. It was even posible with the use of a void function, which does not return any value. This was achieved through the use of pointers. Normally, a copy of an object is sent to the function, but in this case the function obtains a specific address of the object and assigns it to the appropriate pointer. By modifying the pointer, the object value has actually been modified.

1.4 Usage of a pointer when accessing specific memory cells.

It is possible to assign a pointer to a specific memory location. This applies, for example, to data sent from a sensor or other external device to a given address, and this cell is not called in the program. Suppose we want the pointer to point to cell 45634. To assign a pointer to this adress, simply:

```
wsk = 45634;
```

Unfortunately, there are situations where assigning a pointer to a particular address is not so easy. It all depends on the type of computer or operating system. For information on this, see the documentation for specific compilers.

2 Dynamic memory allocation.

2.1 Reservation of memory areas.

To reserve memory areas (eg to create a dynamic array) we will use the **new** operator (in c, the equivalent of malloc). To clear the area **delete** is used (equivalent to c - free).
Let's say we have a pointer:

```
int *wsk;
```

To create a new object in memory we use the following statement:

```cpp
wsk = new char;
```

Thanks to that we created a new char object in memory. This object has no name, but the wsk pointer has its address stored in memory.
To clear the memory from this object we use:

```cpp
delete wsk;
```

Features of objects created in this way:
– This object exists since creation until removal,
The programmer decides the length of his "life" – This object has no name, we can only manipulate it by pointer indicating to it,
– Such an object does not apply to the standard scope of validity. If there is a pointer to it, we can use it,
– Objects created in this way are dynamic, so they will always accept junk values, remember this and make sure you enter propper values to it.

Example:

```cpp
#include <iostream>
using namespace std;

char * tworzobiekt (void); /* tworzymy tu funkcje wywolywana bez
    argumentow, ktora zwracac bedzie wskaznik typu char */

main() {
    char *wsk1, *wsk2, *wsk3;
    cout << "Teraz nastapi akt stworzenia!\n";
    wsk1 = tworzobiekt();
    wsk2 = tworzobiekt();
    wsk3 = tworzobiekt();
    *wsk1 = 'a';
    *wsk2 = 'b';
    cout << "\nStworzylismy trzy nowe obiekty.\n"
    << "Do dwoch przypisano wartosci: "<< *wsk1 << *wsk2 << endl
    << "W trzecim widzimy same smieci: " << *wsk3 << endl;
    delete wsk1;
    delete wsk2;
    delete wsk3;
return 0;
}
```

```
char * tworzobiekt (void) {
    char *w;
    cout << endl << "Tworze obiekt!" << endl;
    w = new char;
    return w;
}
```

2.2   Dynamic array allocation.

It is possible to create arrays with new operator:

```
int *wsk;
wsk = new int [size];
```

Thanks to the above two lines we have created an array of size **size** (integer value). Operator action new creates a space for the nameless array in memory, and assigns its zero element address to a pointer. Such an array works on identical terms as the objects created in the preceding paragraph.
Example:

```
#include <iostream>
using namespace std;

float srednia (float a, float tab[ ]);

main() {
    int rozm=0, i;
    float *wsk, wynik;
    cout << "Program obliczy srednia arytmetyczna kilku liczb\n";
    cout << "Podaj ile liczb chcesz wprowadzic w celu obliczenia"
    << " sredniej: ";
    cin >> rozm;
    wsk = new float[rozm];
    for (i = 0; i<rozm; i++) {
        cout << "Podaj wartosc dla " << i+1 << " elementu: ";
        cin >> wsk[i];
        cout << endl;
    }
    cout << "Policze srednia dla nastepujacych liczb: ";
    for (i=0; i<rozm; i++) {
        cout << wsk[i] << ", ";
    }
```

```
    cout << endl;
    wynik = srednia(rozm, wsk);
    cout << "Srednia powyzszych liczb wynosi: " << wynik << endl;
    delete wsk;
    wsk = NULL;
return 0;
}

float srednia (float a, float tab[ ]) {
    int b;
    float sred;
    for (b =0; b<a; b++) {
        sred += tab[b];
    }
    sred /= a;
return sred;
}
```

Sometimes written programs may consume too much memory and unfortunately will not be able to create a new object. Instead, it will be assigned NULL (0) value. If you plan to create multiple dynamic objects in your program, there is a way to simply check if an object was created. Here is an example of a checking loop:

```
int rozm;
float *wsk;
wsk = new float[rozm];
if (!wsk) {
   error("pamiec sie wyczerpala");
}
```

Task

Based on the informations provided in this manual, please improve the simple utility interface of the snack and beverage vending machine created on previous labratories.

Program requirements:
1. Program should "accept" payment for products and "give" the rest of the money backt to user.
2. The cash pool in the slot machine should be pre-determined during the

initial launch of the program (eg 50PLN). The pool should be divided into specific coin denominations (5,2,1,0.5,0.2,0.1 PLN). Any modifications in the cash pool should be exported to an external file after each transaction.

3. When giving back the rest, the slot machine should set the priority of issuing the coins it has in excess.

4. When a user pays a fee, he / she must enter what coin nominal values are used to pay for the products. If the machine will not be able to spend the rest, you will receive a message asking for payment deducted in cash.

5. Any modification of the amount of coins in the machine will be carried out using pointers.

6. The program should be equipped with a hidden function that runs when the appropriate code is provided, which will allow editing of the money pool available in the machine.