Sortowanie, funkcje

Dziś napiszemy klika algorytmów, potem porozmawiamy o funkcjach, a potem będzie koniec zajęć.

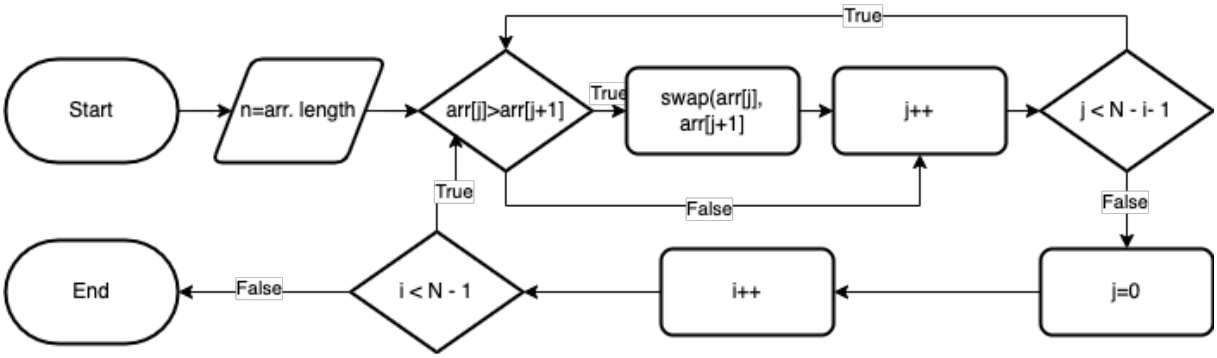
Algorytmy sortujące

Algorytmy sortujące są używane do uporządkowania zadanej tablicy według przyjętego kryterium. Dziś będziemy jedynie poruszali temat implementacji algorytmów, jedynie pokrótce wspominając o ich złożoności, bo to jest temat nawet nie na oddzielne zajęcia, tylko w ogóle inny przedmiot.

Bubble sort (sortowanie bąbelkowe)

W zasadzie najprostszy z algorytmów sortujących, działający na bardzo prostej zasadzie, ale równocześnie najmniej wydajny czasowo i pamięciowo oraz nienadający się do pracy na dużych zbiorach danych. Jednak gdy cenny jest tylko nasz czas, a celem nauka programowania, to jakoś ujdzie.

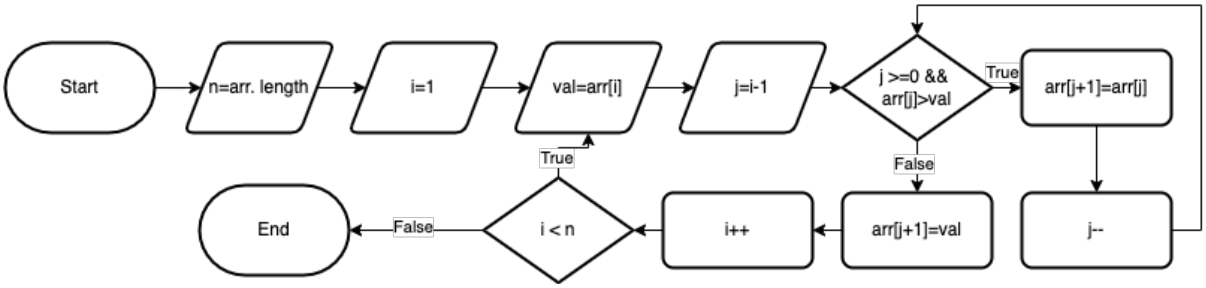
Algorytm sortowania bąbelkowego przechodzi wielokrotnie, element po elemencie przez całą tablicę i zamienia sąsiadujące ze sobą elementy, jeśli jeden jest większy od drugiego.



Ten algorytm zaimplementujemy wspólnie. Nie nadpisujcie go, przyda nam się jeszcze.

Insertion sort (sortowanie przez wstawianie)

Algorytm mocno zainspirowany tym, jak układamy karty na ręce. Tablica jest „wirtualnie” podzielona na część posortowaną i nieposortowaną. Wartości z części nieposortowanej są kolejno brane i układane w odpowiednie miejsce w części posortowanej.



Ten już do zaimplementowania samodzielnego. Warto zwrócić uwagę, że `arr[j+1]=val` zawiera odwołanie do wartości j poza pętlą, więc nie możemy skorzystać tu ze zwykłej pętli for ze zmienną j deklarowaną wewnątrz.

Na chwilę zmiana tematu, funkcje

Funkcja, a właściwie podprogram, to wydzielona część programu, która może, choć nie musi przyjmować jakąś liczbę argumentów, przetwarza je w określony wewnątrz sposób, oraz ewentualnie zwraca jakąś wartość, którą możemy na przykład przypisać do zmiennej albo podać jako argument innej funkcji. Wykorzystanie funkcji powoduje, że możemy zgrupować instrukcje, tworząc z nich bardziej złożone. W efekcie otrzymujemy nową, napisaną pod własne potrzeby „instrukcję", którą możemy wykorzystywać wewnątrz programu wielokrotnie, co w (kolejnym) efekcie pozwoli nam na uniknięcie powtarzania kawałków kodu.

```
typ_funkcji nazwa_funkcji(typ_arg1 nazwa_arg1,typ_arg_n nazwa_arg_n){
      //zawartość funkcji
      return wartosc //zaleznie od typu, nie musi być
}
```

Każda funkcja musi mieć swój typ. Zwykle będzie to typ zwracanych przezeń danych, jednak czasami możemy potrzebować funkcji, która nie będzie sama w sobie zwracała żadnych wartości, tylko coś wykona, np. wypisze coś na ekranie. Wtedy taka funkcja będzie miała typ void.

Ważna uwaga: int main() również jest funkcją, wykonywaną automatycznie przy uruchomieniu programu. Poprawnie napisany program na końcu będzie miał wartość przez program zwracaną, w naszym wypadku zawsze był to 0. Wszystkie inne funkcje mogą być wewnątrz tego maina wywoływane, jednak deklarowane i definiowane muszą być poza nim.

```
int sq_field(int a){
    return a=a*a;
}
```

Powyższa funkcja typu int przyjmuje jeden argument, będzie to liczba całkowita, nazwana a. Do tej zmiennej mamy dostęp tylko wewnątrz danej funkcji. Powyżej prezentowany przykład to tzw. definicja funkcji. Funkcyjka ta obliczy nam kwadrat podanej do niej wartości i go zwróci jako rezultat.

```
int sq_field(int a){
    return a=a*a;
}
int main(){
    int a = sq_field(7);
    cout<<a;
    return 0;
}
```

W powyższym przykładzie możemy zobaczyć, że wartość zwracanej funkcji możemy przypisać do zmiennej. Tu kolejna ważna uwaga, ważna jest kolejność! Nie możemy wywołać żadnej funkcji w mainie, jeśli nie jest ona zadeklarowana lub zdefiniowana przed nią. A o co chodzi z tą deklaracją? Czasami może zdarzyć się sytuacja, w której w definicji jednej funkcji musimy umieścić wywołanie innej. Przy większej ilości takich funkcji napotkać możemy na

problem z odpowiednim ich ułożeniem, dlatego powinniśmy zawsze korzystać z deklaracji. Deklaracja funkcji działa na tej samej zasadzie, co deklaracja zmiennej.

```
int sq_field(int a); //deklaracja

int sq_field(int a){ //definicja
    return a=a*a;
}
```

W programowniu przyjęła się zasada, że na początku programu będziemy deklarować funkcje, później umieścimy funkcję główną, a za nią znajdą się definicje naszych funkcji:

```
int sq_field(int a); //deklaracja

int main(){
    int a = sq_field(7);
    cout<<a;
    return 0;
}
int sq_field(int a){ //definicja
    return a=a*a;
}
```

Każda funkcja może przyjmować wiele argumentów, niekoniecznie tego samego typu. Mogą też nie przyjmować żadnego argumentu. Dla przykładu funkcja, która wyliczy zadaną potęgę zadanej liczby.

```
double power(double a, int n);

int main(){
    cout<<power(3,3)<<" "<<power(2.4, 2);
    return 0;
}
double power(double a, int n){
    double result = 1;
    for (int i=1; i<=n;i++){
        result*=a;
    }
    return result;
}
```

Funkcje, które nie zwracają żadnej wartości będą miały typ void. Mogą one przymować argumenty, ale nie zwrócą żadnej wartości, czyli nie możemy ich do niczego przypisać, ale oczywiście możemy bez problemu je w naszym programie wywoływać. Przy okazji: podawanie tablic do funkcji:

```
void arr_print(int arr[], int size);

int main(){
    int arr[5] = {12, 14, 2, 5, 11};
    arr_print(arr, sizeof(arr)/sizeof(int));
    return 0;
}
void arr_print(int arr[], int size){
    cout<<"{";
    for (int i=0; i<size; i++){
        if (i<size-1) cout<<arr[i]<<", ";
        else cout<<arr[i];
    }
    cout<<"}"<<endl;
}
```

Zadanie na zajęcia, niepunktowane

Przepisz przygotowane na początku algorytmy do odrębnych funkcji, tak aby można je było komfortowo wykonać kilka razy w ramach jednego programu.