Instrukcja wyboru – `switch`

1. Słowo kluczowe `switch`.
2. Wyrażenie w nawiasie okrągłym.
3. Początek bloku.
4. Lista wartości (stałych) do porównania z wyrażeniem, każda ze słowem kluczowym `case` (np. `case 100:`).
5. Opcjonalnie słowo kluczowe `default` oznaczające domyślną akcję.
6. Koniec bloku.

- Jeżeli wartość wyrażenia jest równa wartości za słowem kluczowym `case`, to następuje przeskok do miejsca w bloku, gdzie znajduje się słowo kluczowe `case` z tą wartością.
- W przypadku braku dopasowania następuje przeskok do miejsca w bloku oznaczonego przez `default`.
- `break` powoduje natychmiastowe opuszczenie bloku.

```
int i = 3;
switch (i)
{
  case 1:
    cout << 'a';


  case 5:
    cout << 'e';
  case 3:
    cout << 'c';
  case 4:
    cout << 'd';
  case 2:
    cout << 'b';
}
```

cdb

```
int i = 0;
switch (i)
{
  case 1:
    cout << 'a';


  case 5:
    cout << 'e';
  case 3:
    cout << 'c';
  case 4:
    cout << 'd';
  case 2:
    cout << 'b';
}
```

nic nie wypisze

```
int i = 0;
switch (i)
{
  case 1:
    cout << 'a';
  default:
    cout << '*';
  case 5:
    cout << 'e';
  case 3:
    cout << 'c';
  case 4:
    cout << 'd';
  case 2:
    cout << 'b';
}
```

*ecdb

```
int i = 3;
switch (i)
{
  case 1:
    cout << 'a';
    break;
  default:
    cout << '*';
    break;
  case 5:
    cout << 'e';
    break;
  case 3:
    cout << 'c';
    break;
  case 4:
    cout << 'd';
    break;
  case 2:
    cout << 'b';
    break;
}
```

```
int i = 3;


if (i==1)
  cout << 'a';

else if (i==2)
  cout << 'b';

else if (i==3)
  cout << 'c';

else if (i==4)
  cout << 'd';

else if (i==5)
  cout << '5';

else
  cout << '*';
```

```
double delta = b * b - 4 * a * c;
if ( delta > 0 )
   ...
else if ( delta < 0 )
   ...
else
   ...
```

switch - przykład

```
switch (req) {
case RPM_REQ_NONE:
    break;
case RPM_REQ_IDLE:
    rpm_idle(dev, RPM_NOWAIT);
    break;
case RPM_REQ_SUSPEND:
    rpm_suspend(dev, RPM_NOWAIT);
    break;
case RPM_REQ_AUTOSUSPEND:
    rpm_suspend(dev, RPM_NOWAIT | RPM_AUTO);
    break;
case RPM_REQ_RESUME:
    rpm_resume(dev, RPM_NOWAIT);
}
```

Skoki (*ang. jump*) – goto

```
    suma = a_k = 1;
    for (k = 0; k < N; suma += a_k) {
        if (a_k < epsilon)
            goto dalej;
        a_k /= ++k;
    }
dalej:
```

`dalej:` jest etykietą (*ang. label*) oznaczającą miejsce, od którego należy kontynuować wykonywanie programu.

Instrukcji `goto` nie można używać do „przeskoków" między funkcjami (tzn. docelowa etykieta musi być w treści tej samej funkcji, w której jest instrukcja `goto` prowadząca do niej).

```
while ( w1 )
{
   ...
   if ( w2 )
      break;
   ...
}
```

```
while ( w1 )
{
   ...
   if ( w2 )
      goto etykieta;
   ...
}
etykieta:
```

```
while ( w1 )
{
   ...
  while ( w2 )
  {
     ...
    if ( w3 )
      break;
     ...
  }
  ...
}
```

```
while ( w1 )
{
   ...
  while ( w2 )
  {
     ...
    if ( w3 )
      goto etykieta;
     ...
  }
  etykieta:
  ...
}
```

```
while ( w1 )
{
   ...
   while ( w2 )
   {
      ...
      if ( w3 )
         goto etykieta;
      ...
   }
   ...
}
etykieta:
```

```cpp
#include <iostream>
#include <complex>

using namespace std;

int main()
{
   cout << "Hello";
   goto tutaj;
   cout << "world";
   int i;
   i = 10;
tutaj:
   cout << "!" << endl;
   cout << i << endl;
   return 0;
}
```

```
Hello!
-1881115996
```

```cpp
#include <iostream>
#include <complex>

using namespace std;

int main()
{
   cout << "Hello";
   goto tutaj;
   cout << "world";
   complex<double> i;
   i = 10;
tutaj:
   cout << "!" << endl;
   cout << i << endl;
   return 0;
}
```

← Błąd kompilacji

```
In function 'int main()':
error: jump to label 'tutaj'
error:   from here
error:   crosses initialization of
'std::complex<double> i'
```

Łączenie wyrażeń – , (przecinek)

, (przecinek)

1. Oblicz wyrażenie (lub wykonaj instrukcję) po lewej stronie przecinka (łącznie ze wszystkimi efektami ubocznymi).
2. Zaniedbaj wynik obliczonego wyrażenia.
3. Oblicz wyrażenie (lub wykonaj instrukcję) po prawej stronie przecinka.

Przykład

Następujący kod spowoduje wydrukowanie liczb 20 i 30 (w tej kolejności):

```
int i, b = 20, c= 30;
i = b, c;
cout << i << endl;
i = (b, c);
cout << i << endl;
```

Definiowanie funkcji z argumentami

Argumenty funkcji

1. Definiuje się w nagłówku funkcji, w nawiasie okrągłym za nazwą.
2. Ich definicje rozdziela się przecinkami.
3. Są definiowane podobnie, jak zmienne wewnątrz funkcji.
4. Mogą być wykorzystywane jako zmienne wewnątrz funkcji.

Przykład – funkcja z jednym argumentem

```
double f(double x)
{
    return x*x + 1;
}
```

Zwraca wynik będący kwadratem jej argumentu zwiększonym o 1.

Wywołanie funkcji (*ang. function call*)

Na poziomie kodu źródłowego

Polega na umieszczeniu w instrukcji nazwy funkcji z listą wartości parametrów w nawiasie okrągłym.

Podczas wykonywania programu

Kod (tzn. rozkazy dla procesora) wygenerowany na podstawie instrukcji zawartych w treści funkcji jest wykonywany w punkcie programu, w którym w kodzie źródłowym była umieszczona nazwa funkcji.

Podczas wykonywania programu

Wynik generowany przez funkcję jest wykorzystywany w wyrażeniu w tym miejscu, w którym następuje wywołanie funkcji. Za każdym razem wygenerowanie wyniku następuje w efekcie wykonania kodu funkcji (kodu utworzonego na podstawie instrukcji zapisanych w treści funkcji).

Przekazywanie wartości argumentów do funkcji

Podczas wykonywania programu

1. Tworzone są zmienne, których nazwy i typy danych odpowiadają nazwom i typom danych argumentów funkcji.
2. Wartości znajdujące się na pozycjach odpowiadających argumentom w zapisie wywołania funkcji (np. `suma += sin(a);`) stają się początkowymi wartościami tych zmiennych.
3. Wykonywany jest kod reprezentowany przez instrukcje w treści funkcji, odwołujący się do tych zmiennych.
4. Wykonanie tego kodu może skończyć się wygenerowaniem wyniku, który jest wykorzystywany w sposób określony przez kod wywołujący funkcję.

Przekazywanie wartości argumentów do funkcji – przykład

Całka trapezowa dla funkcji `f()` z poprzedniego przykładu.

```
double trapez(double a, double b, int N)
{
    double delta, suma, x_j;
    int j;

    suma = (f(a) + f(b)) / 2;
    delta = (b - a) / N;
    for (j = 1, x_j = a; j < N; j++) {
        x_j += delta;
        suma += f(x_j);
    }
    suma *= delta;
    return suma;
}
```

Funkcje biblioteczne (*ang. library function*)

„Gotowe" funkcje, które zostały napisane oraz skompilowane przez kogoś innego i znajdują się w specjalnych zbiorach funkcji, zwanych bibliotekami (*ang. library*).

Przykłady

`sin()`, `cos()`, `exp()`, `sqrt()` (pierwiastek kwadratowy).

Pliki nagłówkowe (*ang. header files*)

Pliki zawierające, między innymi, nagłówki funkcji bibliotecznych, które mogą być wykorzystane w programie. Na podstawie nagłówków kompilator znajduje odpowiednie funkcje w bibliotekach.

Nagłówki funkcji „matematycznych" znajdują się w pliku nagłówkowym `cmath`.

Stałe (*ang. constant*)

const

Słowo kluczowe używane przy definiowaniu stałych. Po nim podaje się typ danych, nazwę oraz wartość stałej.

Przykład

```
const double pi = 3.14159265358979323846;
```

W C++ stałe mają własności analogiczne do zmiennych poza tym, że ich wartości nie zmieniają się.

Argument funkcji może być zadeklarowany jako stała i wtedy jego wartość nie może być mdyfikowana wewnątrz funkcji (otrzymuje on wartość przy wywoływaniu funkcji).

Referencje (*ang. reference*)

Referencja jest symbolem definiowanym podobnie, jak zmienna, ale w jej definicji, bezpośrednio przed nazwą, znajduję się znak & (*ang. ampersand*).

Przykład

```
double &ref = x;
```

Wartością referencji jest zmienna odpowiedniego typu, wskazana w jej definicji (np. `x` powyżej).

Referencję można traktować jako alternatywną nazwę (alias) zmiennej.

Referencje i argumenty funkcji

Argument funkcji może być zadeklarowany jako referencja i wtedy:

1. W wywołaniu funkcji na pozycji odpowiadającej temu argumentowi musi stać nazwa zmiennej odpowiedniego typu (nie może to być wyrażenie).
2. Zmienna ta będzie używana w treści funkcji bezpośrednio w miejscach, w których następują odwołania do reprezentującego ją argumentu.

Przykład

```
double oblicz(double r, int &n)
{
    return r / ++n;
}
```

Referencje i argumenty funkcji – przykład

Wywołanie funkcji `oblicz()`

```
k = 0;
a_k = 1;
suma = 1;
while (k < N) {
    a_k = oblicz(a_k, k);
    suma += a_k;
    if (a_k < epsilon)
        break;
}
```

Zmienna `k` jest modyfikowana przez funkcję `oblicz()` (występuje w niej pod nazwą `n`).

Tablice, elementy, indeksy

Tablica (*ang. array*)

Zestaw N zmiennych tego samego typu numerowanych liczbami w zakresie od 0 do $(N-1)$.

Element tablicy

Zmienna wchodząca w skład tablicy, mająca przypisaną określoną liczbę, która ją identyfikuje.

Indeks (*ang. index*) elmentu tablicy

Liczba identyfikująca element tablicy.

Długość tablicy

Liczba elementów tablicy (N).

Rozmieszczenie elementów w pamięci, nazwa tablicy

W C++ zakłada się, że elementy tablicy będą zawsze rozmieszczane w pamięci jeden obok drugiego, zgodnie z kolejnością indeksów (tzn. element o najmniejszym indeksie będzie znajdował się w pamięci w lokacji o najniższym adresie).

Nazwa tablicy

Jest wykorzystywana do identyfikowania tablicy oraz wszystkich jej elementów. W programie reprezentuje adres tablicy, czyli adres jej pierwszego elementu.

`a[j]`

Element tablicy o nazwie `a`, któremu odpowiada indeks `j` (w ogólności `j` może być stałą lub dowolnie skomplikowanym wyrażeniem).

Deklaracja i definicja tablicy

Deklaracja tablicy

Określenie typu danych dla elementów tablicy, nazwy oraz (nie obowiązkowo) liczby elementów w nawiasie kwadratowym, np.

```
int tab[100];
int liczby[];
```

Definicja tablicy

Deklaracja tablicy, w której jest określona liczba elementów.

Po zdefiniowaniu tablicy każdy jej element może być wykorzystywany jako niezależna zmienna.

```
int liczby[] = { 2, 0, 1, 1 };
```

Tablica – przykład

```
int tab[10];
...
for (int j = 0; j < 10; j++)
    tab[j] = 0;
```

Uwaga!

1. Kompilator C++ nie sprawdza, czy używane w programach wartości indeksów tablic są właściwe.
2. Wartości te również nie są kontrolowane podczas wykonywania programu.
3. Ujemne wartości indeksów są dopuszczalne (oznaczają hipotetyczne elementy o adresach mniejszych od adresu pierwszego elementu).

Przykład zastosowania tablicy

Tablice dobrze sprawdzają się w zastosowaniach, w których mamy do czynienia ze stosunkowo dużą, ustaloną liczbą obiektów tego samego typu.

Problem

W zbiorze $(N + M + 1)$ wylosowanych liczb mamy znaleźć taką, dla której w tym zbiorze jest N liczb nie większych i M nie mniejszych od niej. Dla $N = M$ jest to problem wyznaczania mediany („środkowego elementu") zbioru.

Metoda poszukiwania rozwiązania

Ustawimy liczby w danym zbiorze w kolejności rosnącej i wybierzemy tę, która będzie na pozycji $(N + 1)$.

Algorytm sortowania tablicy (przez wybieranie)

1. Zapisz każdą wylosowaną liczbę w innym elemencie tablicy $a[]$ o długości $L = N + M + 1$.
2. Powtarzaj dla indeksów i od 0 do $(L - 2)$:
   1. Wyznacz indeks j taki, że element tablicy o tym indeksie jest najmniejszy spośród elementów o indeksach od i do $(L - 1)$ włącznie.
   2. Zamień miejscami element o indeksie i z elementem o indeksie j.
3. Po zakończeniu powyższych czynności tablica jest posortowana.

Wyznaczanie najmniejszego elementu (krok 2.1)

1. Niech $j = i$.
2. Powtarzaj dla indeksów k od $(i + 1)$ do $(L - 1)$:
   - Jeśli $a[k] < a[j]$, to niech $j = k$.
3. j jest poszukiwanym indeksem.

Porządkowanie przez wybór

(7,1,5,8,9,4,2)
(1)(7,5,8,9,4,2)
(1,2)(7,5,8,9,4)
(1,2,4)(7,5,8,9)
(1,2,4,5)(7,8,9)
(1,2,4,5,7)(8,9)
(1,2,4,5,7,8)(9)
(1,2,4,5,7,8,9)

n^2

Program sortujący tablicę

```
for (int i = 0; i < L-1; i++) {
    // Poszukiwanie indeksu j
    int j = i;
    for (int k = i + 1; k < L; k++)
        if (a[k] < a[j])
            j = k;

    // Zamiana miejscami a[i] z a[j]
    if (j > i) {
        double m = a[j];
        a[j] = a[i];
        a[i] = m;
    }
}
```

Program sortujący tablicę – c. d.

Wypełnianie tablicy

W powyższym kodzie źródłowym pominięto wypełnianie tablicy:

```
srandom(time(NULL));
for (int i = 0; i < L; i++)
    a[i] = 1.0 / (1.0 + random());
```

Funkcja `random()` generuje liczby pseudolosowe w zakresie od 0 do `RAND_MAX`. Do tablicy wstawiane są liczby pseudolosowe z przedziału $(0, 1]$.

Wynik obliczeń

Po przeprowadzeniu sortowania tablicy wystarczy wydrukować $a[N]$ jako wynik.

Sito Eratostenesa

Poszukujemy liczb pierwszych nie większych od N

Niech $A = \{2, 3, ..., N\}$ będzie zbiorem liczb, natomiast k i m – miejscami do przechowywania pośrednich wyników.

1. Niech $k = 2$.
2. Jeśli $k^2 > N$, zbiór A zawiera tylko liczby pierwsze.
3. Usuwamy z A wszystkie wielokrotności k, począwszy od k^2.
4. W m zapisz najmniejszą liczbą ze zbioru A większą od k.
5. Niech $k = m$.
6. Przejdź do kroku 2.

Zbiór A w powyższym algorytmie można zastąpić tablicą.

Sito Erastotenesa wykreślanie liczb *i*j*

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	100

Sito Eratostenesa z użyciem tablicy

Poszukujemy liczb pierwszych nie większych od N

Niech $a[]$ będzie tablicą o $(N+1)$ elementach typu `bool`.

1. Niech wszystkie elementy $a[j]$ dla $j > 1$ mają wartość `true`.
2. Niech $k = 2$
3. Wstaw `false` do wszystkich elementów $a[j]$, dla których j jest wielokrotnością k, począwszy od k^2.
4. Powtarzaj:
   - $k = k + 1$
   - Jeśli $k^2 > N$, przejdź do kroku 5.
   - Jeśli $a[k]$ ma wartość `true`, przejdź do kroku 3.
5. Wydrukuj indeksy j, dla których elementy $a[j]$ mają wartość `true`.

Sito Eratostenesa z użyciem tablicy – program

```
bool a[N+1];
int j;

for (j = 2; j <= N; j++)
    a[j] = true;

for (int k = 2; k*k <= N; k++)
    if (a[k]) {
        for (j = k*k; j <= N; j += k)
            a[j] = false;
    }

for (j = 2; j <= N; j++)
    if (a[j])
        cout << j << endl;
```

Sito Eratostenesa z użyciem tablicy – usprawnione

W poprzednim programie nie wykorzystujemy elementów `a[0]` i `a[1]`.

Poszukujemy liczb pierwszych nie większych od N

Niech $a[]$ będzie tablicą o $(N-1)$ elementach typu `bool`.

1. Niech wszystkie elementy $a[j]$ mają wartość `true`.
2. Niech $k = 2$
3. Wstaw `false` do wszystkich elementów $a[j]$, dla których $(j+2)$ jest wielokrotnością k, począwszy od k^2.
4. Powtarzaj:
   - $k = k + 1$
   - Jeśli $k^2 > N$, przejdź do kroku 5.
   - Jeśli $a[k-2]$ ma wartość `true`, przejdź do kroku 3.
5. Wydrukuj liczby j, dla których elementy $a[j-2]$ mają wartość `true`.

Sito Eratostenesa z użyciem tablicy – poprawiony program

```
bool a[N-1];
int j;

for (j = 0; j < N-1; j++)
    a[j] = true;

for (int k = 2; k*k <= N; k++)
    if (a[k-2]) {
        for (j = k*k - 2; j < N-1; j += k)
            a[j] = false;
    }

for (j = 0; j < N-1; j++)
    if (a[j])
        cout << (j + 2) << endl;
```

Sito Eratostenesa z użyciem tablicy i wzorów bitowych

Poszukujemy liczb pierwszych nie większych od N

Niech $a[]$ będzie tablicą o $(N+6)/8$ elementach typu `unsigned char`.

1. Niech wszystkie elementy $a[j]$ mają wartość 0 (wszystkie bity są zerami).
2. Niech $k = 2$
3. Wstaw 1 do wszystkich bitów w $a[]$ odpowiadających wielokrotnościom k, począwszy od k^2.
4. Powtarzaj:
   - $k = k + 1$
   - Jeśli $k^2 > N$, przejdź do kroku 5.
   - Jeśli bit w $a[]$ odpowiadający k jest zerem, przejdź do kroku 3.
5. Wydrukuj liczby j, dla których odpowiadające im bity w $a[]$ są zerami.

Sito Eratostenesa z użyciem tablicy i wzorów bitowych c. d.

Bit w $a[]$ odpowiadający liczbie k

1. Pierwszy bit odpowiada liczbie 2.
2. Każdy element $a[]$ odpowiada 8 bitom.
3. Indeks: $i = (k-2)/8$ (dzielenie z pominięciem reszty).
4. Pozycja bitowa: $b = (k-2)\,\%\,8$.

Sprawdzanie wartości

```
(a[(k-2)/8] & (1 << ((k-2) % 8)) == 0
```

Wstawianie jedynki

```
a[(k-2)/8] |= 1 << ((k-2) % 8)
```

Sito z użyciem tablicy i wzorów bitowych – program

```
unsigned char a[(N+6)/8];
int j;

for (j = 0; j < (N+6)/8; j++)
    a[j] = 0;

for (int k = 2; k*k <= N; k++)
    if ((a[(k-2)/8] & (1 << ((k-2) % 8))) == 0) {
        for (j = k*k; j <= N; j += k)
            a[(j-2)/8] |= 1 << ((j-2) % 8);
    }

for (j = 2; j <= N; j++)
    if ((a[(j-2)/8] & (1 << ((j-2) % 8))) == 0)
        cout << j << endl;
```